CEKIT S.A.

Curso práctico sobre

MICROCONTROLADORES

Teoría, Programación, Diseño, Prácticas y Proyectos completos

Teoría

Los microprocesadores
Evolución histórica

Programación

Conceptos básicos

Lenguajes de programación

Proyectos

Introducción

Pasos a seguir para elaborar un proyecto

Argentina: $ 3.90
Colombia: $ 5.500
Chile: $ 1.450
México: $ 27
Venezuela: Bs. 2.000

Curso práctico sobre

MICROCONTROLADORES

Teoría, Programación, Diseño, Prácticas y Proyectos completos

El **Curso práctico sobre MICROCONTROLADORES** de CEKIT S. A. se publica en forma de 40 fascículos de aparición semanal, encuadernables en 3 volúmenes. Cada fascículo consta de 4 páginas de cubiertas y 20 páginas de contenido a todo color. De estas últimas, 8 están dedicadas a la sección de **TEORIA**, 4 a **PROGRAMACIÓN** y 8 a la sección de **PROYECTOS.**

Las páginas de cada sección son encuadernables por separado. Por esta razón, al formar los volúmenes, debe cuidarse de desprender previamente, de todos los fascículos, las 8 páginas centrales de **PROYECTOS**, las 4 de **PROGRAMACIÓN** y por último las 8 páginas de **TEORÍA**. Durante la publicación de los fascículos se pondrán a la venta las tapas de los 3 volúmenes de la obra para su encuadernación.

CEKIT S.A. garantiza la publicación de la totalidad de la obra, el suministro de la tapas necesarias para su encuadernación y el servicio de números atrasados hasta un año después de terminada la circulación del último fascículo. También garantiza la fidelidad de la información teórico-práctica suministrada y el correcto funcionamiento de los programas incluidos como ejemplos de la obra.

Motorola, Microchip, PIC, Intel, Atmel, Parallax, Zilog, Phillips, son marcas registradas de las diferentes empresas en los Estados Unidos y en otros paises. Otros productos y nombres de compañías mencionadas en esta obra pueden ser marcas registradas de sus propietarios.

Este curso ha sido elaborado según el plan del editor y de sus autores, y bajo su responsabilidad, por los siguientes integrantes del departamento técnico de CEKIT S.A.

Distribución Internacional

Argentina: Importador: Editorial Conosur S.A. Av. Belgrano 355 10º. Tel: 4342-9029 Fax: 4342-9025 e-mail: gconosur@speedy.com.ar Dist. Capital: VACCARO SANCHES Y CIA. - Moreno 794, 9º(1092) Buenos Aires. Dist. Interior: DISTRIBUIDORA BERTRAN S.A.C. - Av. Vélez Sarsfield 1950(1285) Buenos Aires. - Bolivia: AGENCIA MODERNA LTDA. - Chile: DISTRIBUIDORA ALFA S.A. - Colombia: DISTRIBUIDORAS UNIDAS. - Venezuela: DISTRIBUIDORA CONTINENTAL.- México: CONSORCIO SAYROLS - Ecuador: DISTRIBUIDORA ANDES. - Panamá: PANAMEX S.A. - Perú: DISTRIBUIDORA BOLIVARIANA S.A. - Paraguay: SELECCIONES S.A.C. - Uruguay:DISTRIBUIDORA CAREAGA

Suscripciones (no válidas en Argentina), números atrasados y consultas técnicas en Colombia

CEKIT S.A.
Calle 22 Nº 8 - 22, Piso 2, A.A. 194 Pereira(Colombia)
o a nuestro correo electrónico:
ecekit@col2.telecom.com.co

ISBN (Fascículo 8):958-657-510-1
ISBN (Volumen 1):958-657-498-9
ISBN (Obra completa): 958-657-497-0

Calle 22 Nº 8 - 22, Piso 2,
Teléfonos
(6) 3348179- 3348187-
3348189 - 3348193
Fax: (6) 3348020
Pereira-Colombia
Carrera 13 Nº 38 - 43, Piso 1.
Teléfonos: 2873086 -
2873420 - 2877318
Bogotá - Colombia

e-mail: ecekit@col2.telecom.com.co
http://www.cekit.com.co

Gerente General:
Jorge Hernán Alvarez
Gerente Comercial:
Marcelo Alvarez H.
Director Editorial:
Manuel Felipe González G.
Circulación Internacional:
Humberto Real Blanco.

Autores

JUAN ANDRÉS CASTAÑO W.
MANUEL FELIPE GONZÁLEZ G.
GILBERTO VARGAS C.
JUAN DIEGO HERNÁNDEZ F.

Proyectos

JOHN JAIME ROBBY G.

Diseño carátula: Luz Angela Plata B.
Diseño: Gloria Marcela Gómez G.
Diagramación: Héctor Hugo Jiménez

© CEKIT S.A. 2001
Pereira - Colombia

Todos los derechos reservados. Prohibida su reproducción parcial o total por cualquier medio sin permiso escrito del editor.

EDICIÓN CONO SUR
Internacional Graphics & Printing Co.
2600 Douglas Road, Suite 406
Coral Gables, Florida 33134, U.S.A.

Impreso y encuadernado por:
RR Donnelley
América Latina
Ruta Panamericana Km. 38.700
Buenos Aires Argentina
Impreso en Argentina
Junio 2002

Curso práctico sobre

MICROCONTROLADORES

Teoría, Programación, Diseño, Prácticas y Proyectos completos

Estudie, con el método de CEKIT de **"Aprender haciendo"**, la tecnología de los Microcontroladores, una de las que más han cambiado el mundo en los últimos años y gane mucho dinero. Estos pequeños cerebros electrónicos se encuentran en todas partes: automóviles, computadoras, electrodomésticos, alarmas, juegos de luces, instrumentos, equipos de comunicaciones, etc.

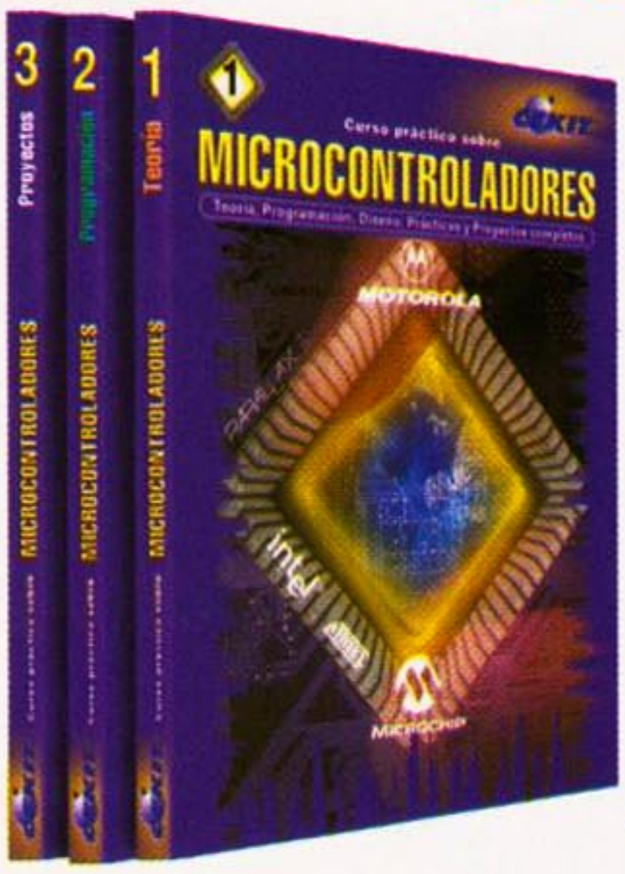
En este curso incluimos los principios teóricos, los fundamentos de su programación, una gran cantidad de actividades prácticas y proyectos completos para ensamblar.

Los microcontroladores son los elementos básicos y fundamentales de una nueva revolución, la interconexión de todo con todo en una red mundial de información y control. Esto permitirá la comunicación de hechos y datos entre personas y objetos formando la red del conocimiento, base de la economía del nuevo milenio **Capacítese y forme parte activa de la nueva economía.**

Libere su creatividad

Las personas con buenos conocimientos de esta tecnología, tendrán muchas oportunidades laborales o podrán fabricar sus propios aparatos en forma industrial, lo que puede ser la base para una empresa muy rentable.

Como está estructurado el curso



El **Curso práctico sobre Microcontoladores** de **CEKIT S.A.** se publicará en 40 fascículos con tres secciones cada uno, y al final, se agrupará en tres tomos de pasta dura con cada uno de los temas principales. Para su fácil identificación, cada una de las secciones tiene un color diferente. La distribución de los tomos es la siguiente:

Tomo 1: Teoría	Tomo 2: Programación	Tomo 3: Proyectos
320 páginas (1 a 320)	160 páginas (1 a 160)	320 páginas (1 a 320)

Sección de Teoría

En la sección de teoría, se estudian los fundamentos teóricos y funcionales de los microcontroladores. Esto se refiere a la estructura o arquitectura interna de los mismos, así como a la función de cada uno de los terminales o pines de los diferentes dispositivos tratados en el curso cuyo conocimiento es necesario para diseñar circuitos con estos componentes.

Se empieza con un repaso sobre el tema de los microprocesadores, los cuales dieron origen a los microcontroladores, y se desarrollan durante las lecciones, los diferentes aspectos técnicos de esta tecnología.

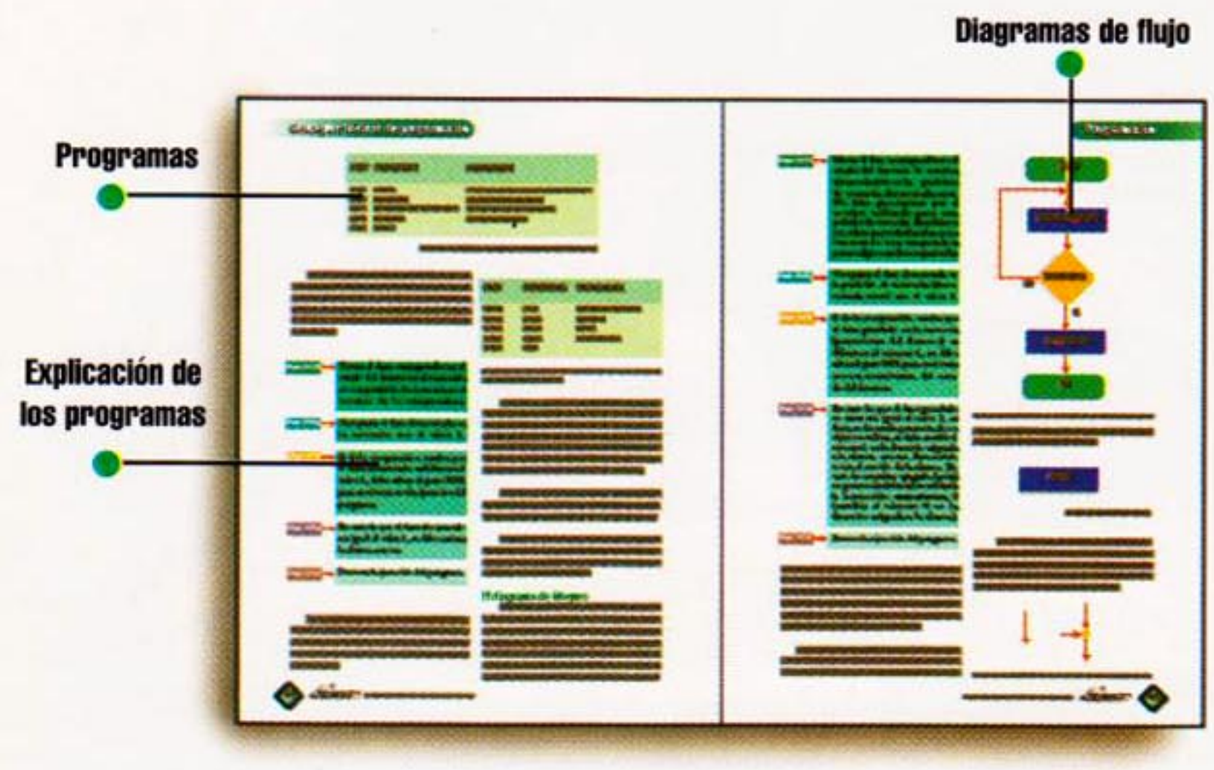
Los siguientes son los principales temas tratados en esta sección:

- Introducción a los microprocesadores
- Introduccion a los microcontroladores
- Diferencias entre un microprocesador y un microcontrolador
- Principales familias de microcontroladores
- Estuctura básica de los sistemas con microcontrolador
- (Motorola y PIC de Microchip)
- Arquitecturas de microcontroladores
- La CPU y la ALU
- Los registros
- Organización de la memoria
- La EPROM de datos
- Las interrupciones
- Los puertos de entrada/salida (I/O)
- Los temporizadores (*Timers*)
- Las opciones Comparación/Captura/PWM
- El puerto serial síncrono SSP, bssp
- La USART
- Las referencias de voltaje
- Los comparadores
- Los conversores A/D
- Modos de bajo consumo (*sleep* y *watchdog*)
- Bits de configuracion
- Programación en el circuito (*in circuit*)
- El conjunto (*set*) de instrucciones
- Las comunicaciones seriales
- Otros microcontroladores
- Parallax (Basic Stamp)
- Intel
- Atmel

Sección de Programación

Todo aparato o proyecto con microcontroladores incluye, además de los diferentes circuitos, uno o varios programas que se deben encribir en el lenguaje apropiado según el tipo de microcontrolador utilizado. Este curso le proporcionará los conocimientos necesarios para que adquiera la habilidad de programarlos.

Los principales temas tratados son:

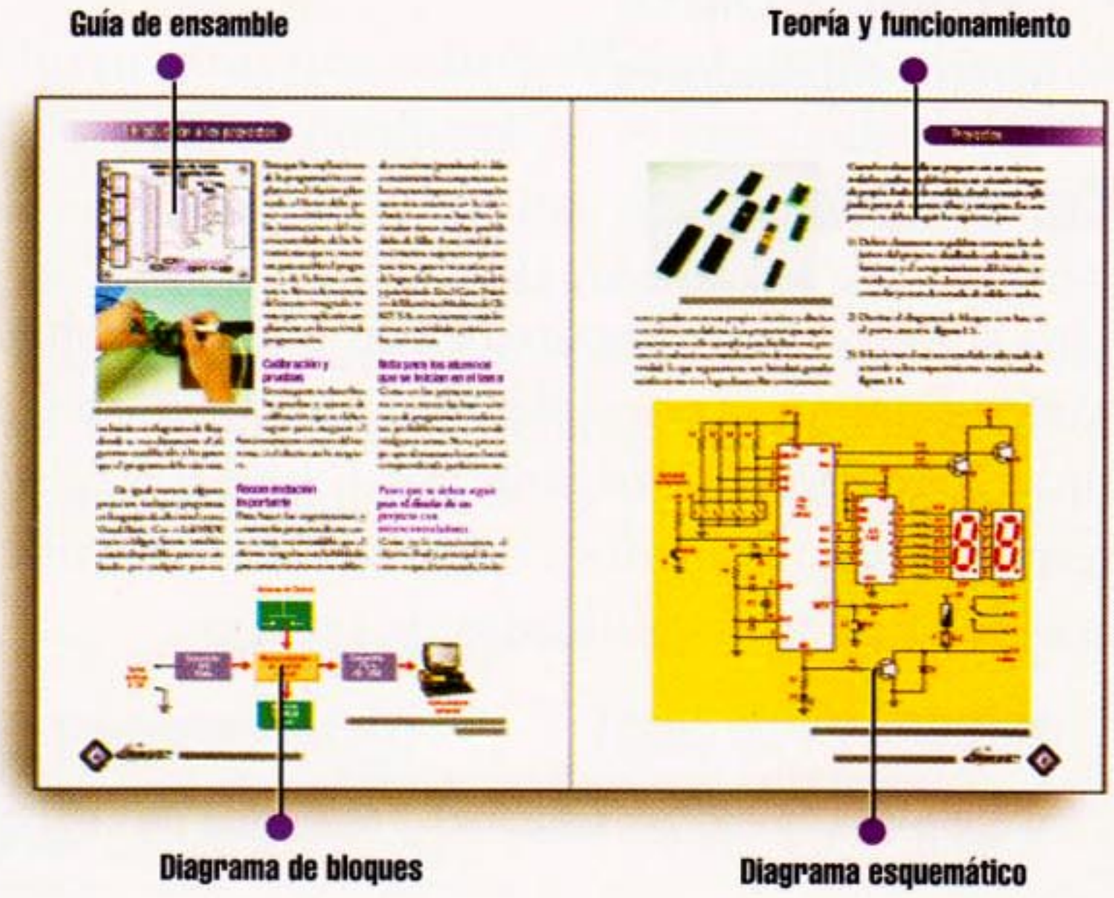


- Programación de un microcontrolador
- El ciclo de desarrollo de un programa
- Herramientas de *software*
- Programación en modo inmediato
- Programación orientada a registros
- Control de flujo de un programa
- Puertos de entrada/salida
- Subrutinas
- Interrupciones
- Programación en lenguaje estructurado
- Temporizadores
- Programación serial
- Programando los conversores A/D y D/A
- Programación en lenguaje C
- Tópicos avanzados de programación

Sección de proyectos

- Indicador binario de 4 bits
- Alarma para el hogar y el automóvil
- Reloj digital y calendario
- Secuenciador de luces programable
- Cerradura codificada
- Activación de aparatos vía telefónica
- Temporizador universal
- Aviso luminoso con diodos LED
- Luces rítmicas
- Convertidor paralelo-serial
- Punta lógica
- Temporizador miniatura
- Frecuencímetro
- Generador de señales
- Programador de memorias
- Adquisicion de datos en 16 bits
- Minirobot con microcontrolador
- Adquisición de datos por el puerto serial
- Tarjeta I/O para el puerto serial
- y muchos otros...

En esta importante sección del curso, se entrega una amplia información con la cual el alumno podrá ensamblar una buena cantidad de proyectos. Para cada uno de ellos se incluye la teoría de funcionamiento, el diagrama esquemático, la lista de materiales, las instrucciones para el ensamblaje y el listado del programa.



A quienes va dirigido el curso

El tema de los microcontroladores es de gran utilidad para un grupo muy amplio de personas que requieran conocer esta tecnología por su gran cantidad de aplicaciones.

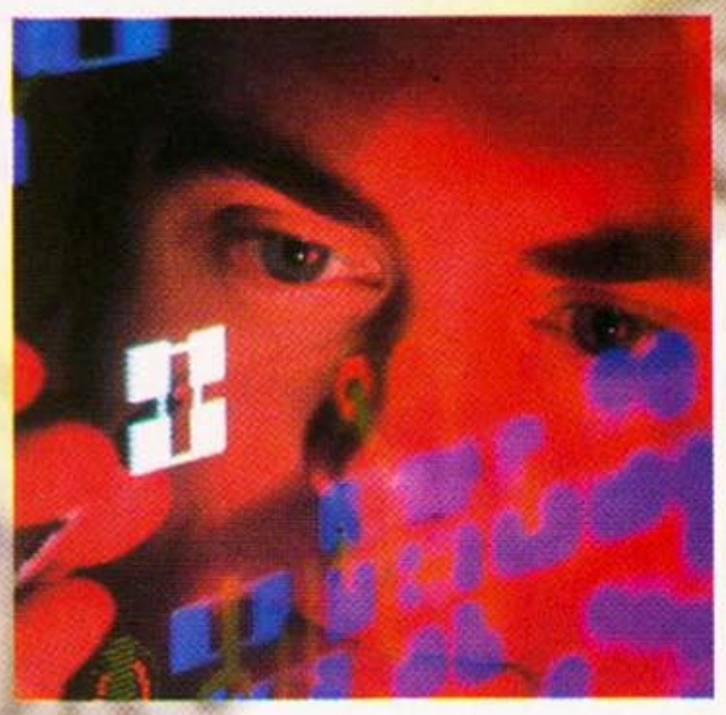
Los principales grupos de usuarios son:

Estudiantes y profesionales de Ingeniería y Tecnologías:

- Electrónica
- Eléctrica
- Mecánica
- Sistemas
- Bioingeniería
- Mecatrónica
- Automatización
- Estudiantes de colegios técnicos con énfasis en electrónica

Técnicos en:

- Electrónica industrial
- Automatización
- Electrónicos en general
- Alumnos de los cursos de **CEKIT S.A.**
- Estudiantes y profesionales en FISICA, para sus necesidades de instrumentación



Autodidactas o aficionados con conocimientos en:

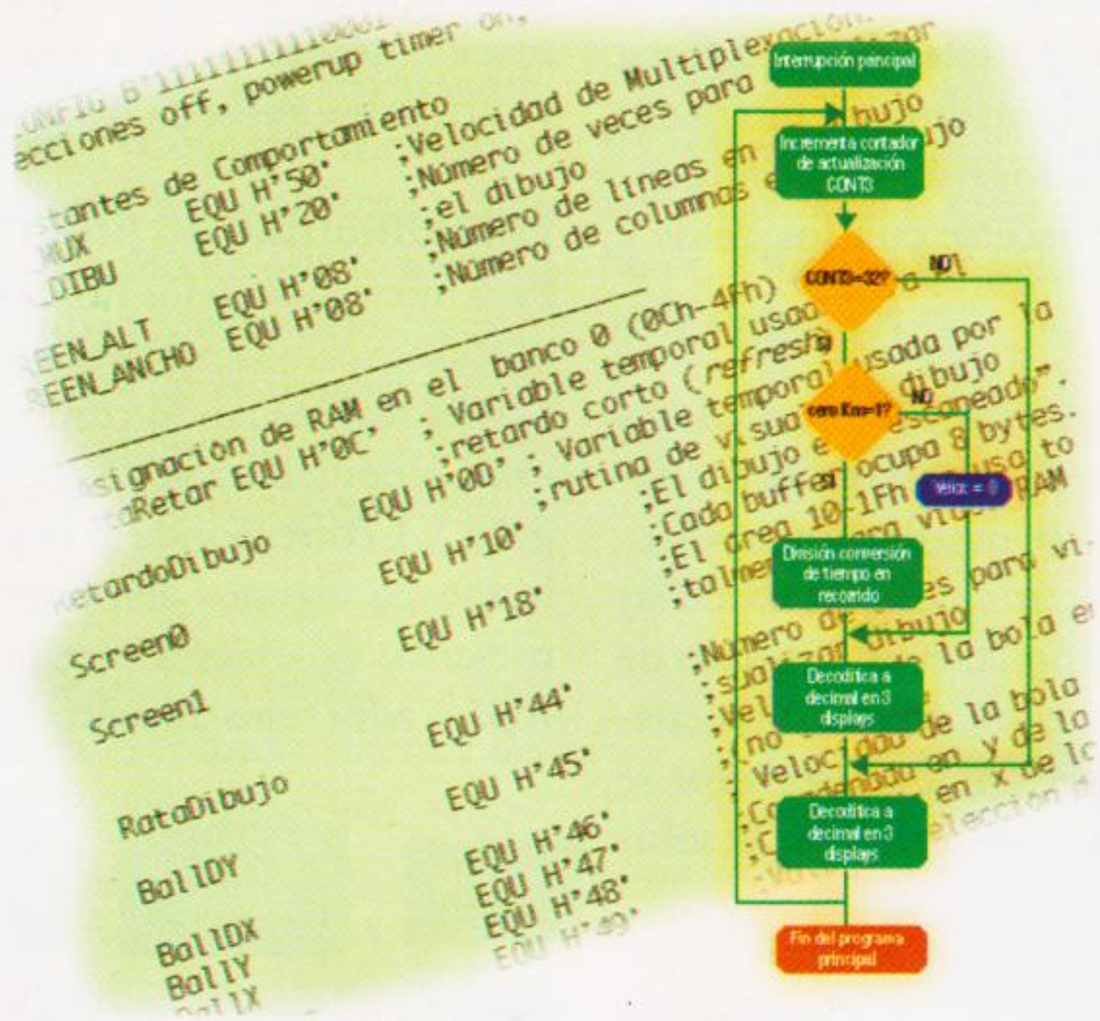
- Electrónica Digital
- Microprocesadores

Certificado

Una vez terminado el curso y aprobada la evaluación que se publicará al finalizar la obra, el alumno recibirá un Certificado de Aptitud Ocupacional otorgado por CEKIT S.A. y aprobado por las respectivas autoridades educativas. Este certificado ler permitirá acceder a puestos calificados de trabajo.

Otra obra con la calidad editorial y didáctica de CEKIT S.A.®
Capacitamos para vivir mejor

LECCIÓN 1



CONCEPTOS BÁSICOS DE PROGRAMACIÓN

Cuando se incluye un microcontrolador en un circuito o en un aparato electrónico, se debe establecer la combinación adecuada de *hardware* (circuitos) y *software* (programas). A esto se le llama un diseño con microcontrolador. Cuando éste se adquiere del fabricante o a través de un distribuidor, no incluye el programa el cual debe ser planeado y escrito por el diseñador del *hardware* o por un programador, que en la mayoría de los casos, es él mismo.

Introducción

Los microcontroladores permiten configurar un sistema de acuerdo con los requerimientos impuestos por el problema que se va a resolver, gracias a una característica fundamental que comparten con las computadoras convencionales: son programables. Por esto es posible incorporarlos a un sistema electrónico, normalmente orientado a la captura y entrega de información, permitiendo decidir lo que debe hacerse con los datos y señales procesados.

El arte de trabajar con microcontroladores requiere del dominio de dos habilidades fundamentales: la primera es la destreza para seleccionar componentes electrónicos y conectarlos adecuadamente, y la segunda, es el dominio de las técnicas de programación, gracias a las cuales se puede lograr que el microcontrolador se comporte según los requerimientos de la solución propuesta.

Si el lector ya ha estado inmerso en el mundo de la electrónica, entonces tendrá las bases necesarias para entender el funcionamiento interno y operativo del microcontrolador y podrá configurar el *hardware* o los circuitos sin mayores dificultades. Sin embargo, el aspecto de la programación puede resultarle un poco complejo. De todas maneras, independientemente de las bases teóricas y prácticas que usted posea, encontrará en este curso los conocimientos de programación necesarios para realizar con éxito los proyectos propuestos en otras secciones de la obra.

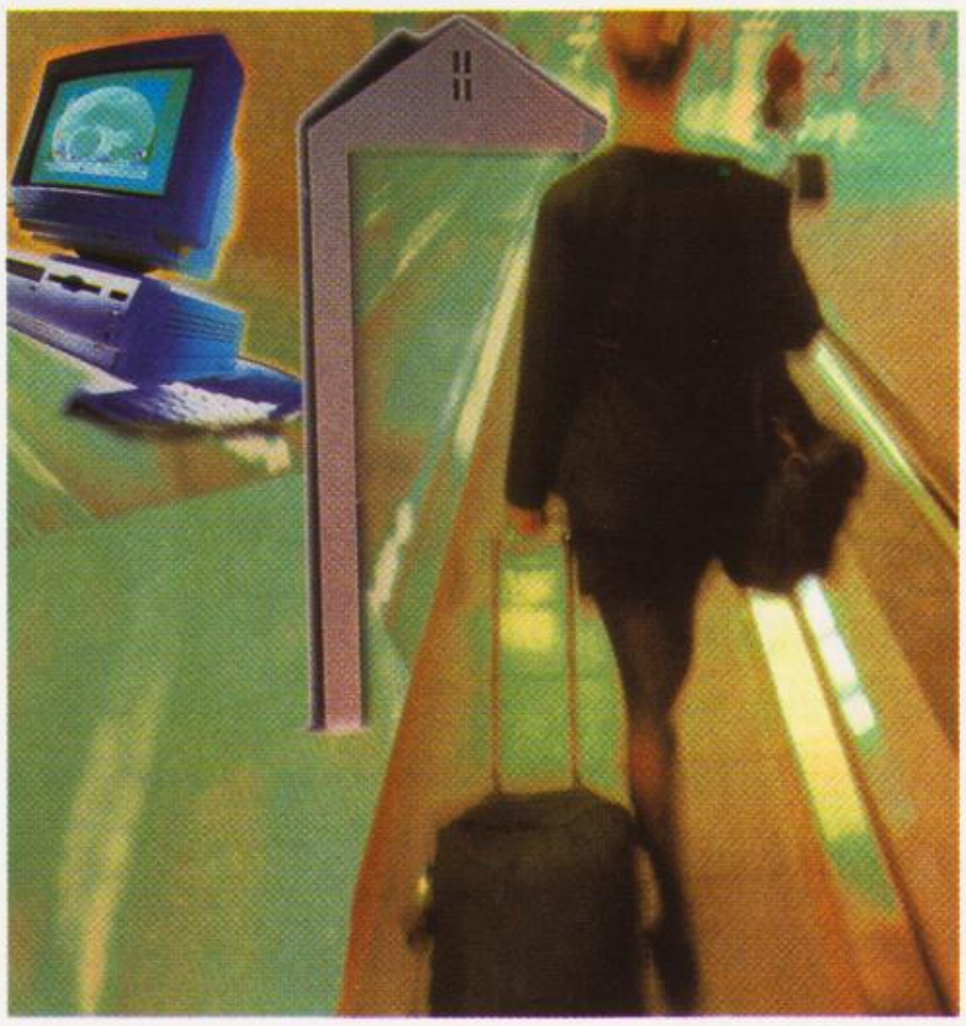


Figura 1.1 *Sistema de control para la detección de objetos metálicos*

Fundamentos de programación

Para iniciarnos en los fundamentos de la programación, veamos primero un sencillo ejemplo de la vida real tomando como elemento de control una computadora genérica. El diagrama de la **figura 1.1** corresponde al esquema de control utilizado en algunos aeropuertos para detectar la presencia de objetos metálicos escondidos en la ropa o adheridos al cuerpo de los pasajeros.

Figura 1.2 *Diagrama de bloques del sistema de detección de objetos metálicos*

La misión del detector de metales es la de producir una señal eléctrica cada vez que una persona cruce frente al detector portando algún objeto metálico. En cuanto a la alarma sonora, ésta debe producir una señal audible en presencia de un elemento peligroso, con el fin de alertar a las autoridades correspondientes, **figura 1.2**.

¿Cuál es el papel que cumple la computadora en este proceso? Como seguramente podrá deducir el lector, la computadora está conectada de modo permanente tanto al detector de metales, como al dispositivo sonoro, **figura 1.3**.

La interfaz de entrada se encarga de adaptar la señal proveniente del detector, introduciéndola en la memoria de la computadora. El amplificador de salida se utiliza para aumentar el tamaño de la señal eléctrica generada, aplicándola a la alarma sonora y generando un sonido perceptible por el oído humano.

Figura 1.3 *Diagrama de bloques del sistema de detección de objetos metálicos*

Ahora surge una pregunta más interesante que la anterior: ¿cómo es posible que la computadora active la alarma sonora únicamente cuando un pasajero cruza portando un objeto metálico, mientras que la mantiene en silencio cuando no detecta el metal? Antes de responder a esta pregunta, es necesario comprender un aspecto clave relacionado con el funcionamiento interno de una computadora.

En contra de lo que pudiera pensarse, ésta no "observa" ni controla el mundo exterior de manera continua. Ella utiliza pequeños períodos de tiempo para realizar las acciones que le son encomendadas, por lo que la única opción que le queda es la de organizar sus tareas de forma secuencial en el tiempo, **figura 1.4**. Este punto debe recordarse siempre, ya que es la base del funcionamiento de cualquier sistema basado en computadoras, microprocesadores o microcontroladores.

Figura 1.4 *La computadora trabaja a intervalos de tiempo predefinidos*

En palabras sencillas, ¿qué funciones debería realizar la computadora de la figura 1.1 para ser eficiente en la tarea de detectar objetos metálicos?. Observe los siguientes pasos presentados en la **figura 1.5**.

Figura 1.5 *Programa para resolver el problema del detector de metales.*

Ahora le preguntamos al lector: ¿en dónde piensa usted que debe almacenarse este conjunto de instrucciones? La respuesta no es complicada: en el interior de la computadora, de un microcontrolador o de un sistema con microprocesador.

Pues bien, el conjunto de instrucciones que acabamos de presentar es lo que se conoce con el

nombre de PROGRAMA y representa el corazón de los sistemas electrónicos modernos, en los cuales es tan importante la interconexión de los circuitos eléctricos como el desarrollo de los programas orientados a la solución del problema. Recuerde que:

Los programas son escritos por PROGRAMADORES

La misión de un programador es la de entender el problema que debe resolver, escribir la secuencia de instrucciones requeridas en el lenguaje apropiado, almacenar la secuencia de instrucciones en la memoria interna de la computadora y, por último, ejecutar o poner en marcha el programa creado, **figura 1.6**.

Figura 1.6 *El trabajo de un programador*

Lenguajes de programación

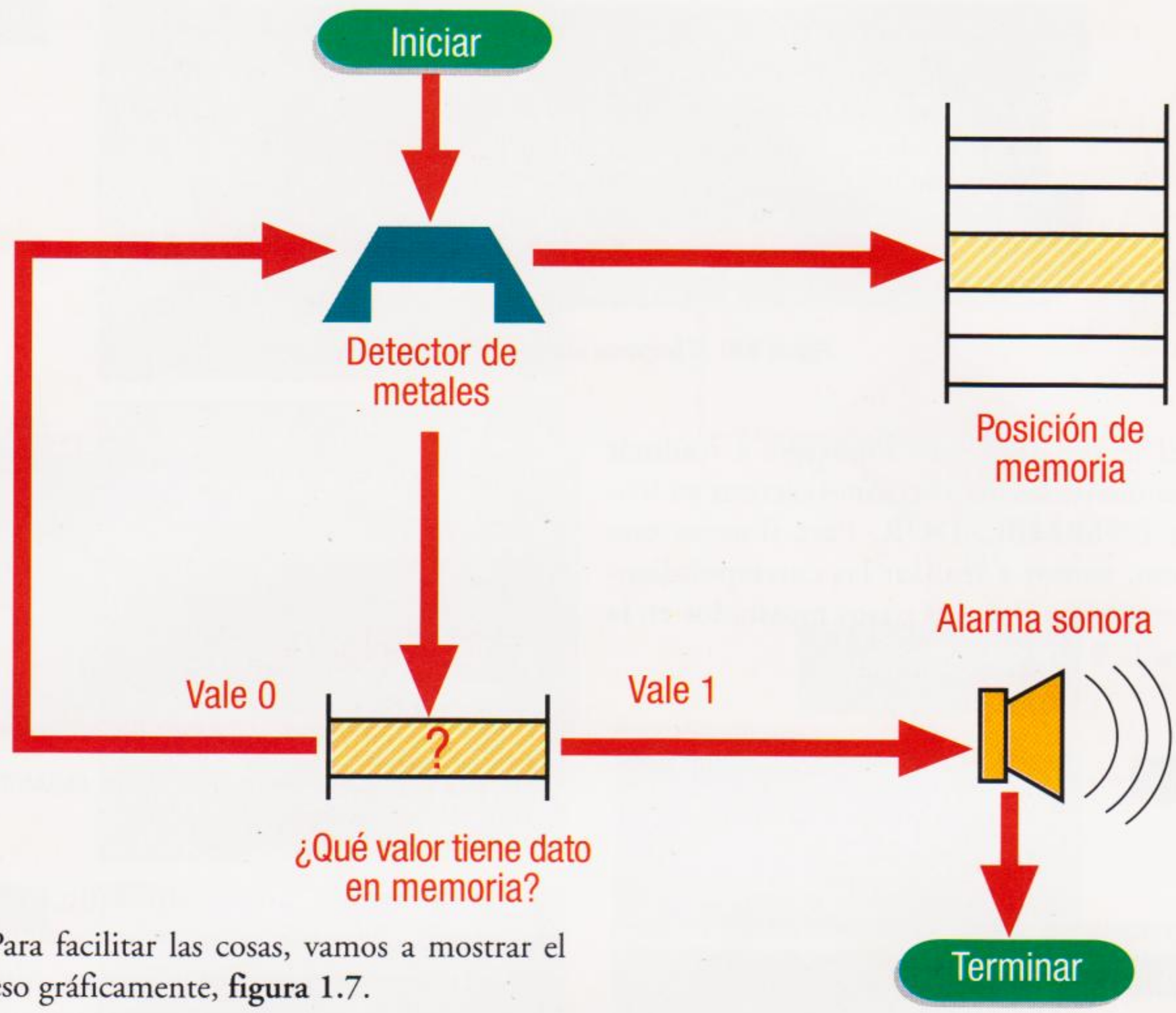
¿Qué lenguaje debe utilizarse para escribir un programa? La respuesta depende de la computadora o en nuestro caso, del microcontrolador utilizado y del problema que se va a resolver. Si su problema es únicamente describir los pasos que deben seguirse para alcanzar el resultado deseado, es suficiente con emplear frases tomadas del idioma español, naturalmente en el orden que resulte más adecuado. Un ejemplo de ello es el programa que presentamos en la figura 1.5.

Pero, ¿será posible que una computadora, como una de las muchas utilizadas en un aeropuerto, pueda entender este tipo de programa? El sentido común nos dice que no: es necesario convertir cada una de las instrucciones al "idioma" que comprenden las computadoras. Existen varios idiomas o lenguajes utilizados por las computadoras modernas. Algunos de ellos tienen como finalidad ayudar a resolver problemas de carácter administrativo, como el lenguaje COBOL, por ejemplo, mientras que otros ayudan a crear programas de utilidad para ingenieros, como FORTRAN o PASCAL, para citar solamente dos ejemplos.

Cuando se trata de resolver un problema cuya característica es el control de componentes electrónicos, como en el caso de los microcontroladores, se debe recurrir a un lenguaje especializado. El más común de ellos recibe el nombre de LENGUAJE ENSAMBLADOR o *ASSEMBLER*. Este lenguaje está compuesto por un conjunto de palabras sencillas que permiten describir acciones básicas, como por ejemplo el movimiento de datos entre las diferentes partes del sistema.

Una de las dificultades que aparece cuando se utiliza el lenguaje ENSAMBLADOR tiene que ver con el hecho de que cada computadora o cada familia de microcontroladores tiene el suyo propio. Sin embargo, y de acuerdo a nuestra experiencia, estamos convencidos de que si usted aprende a programar correctamente desde la base, en un cierto lenguaje, sin importar cual sea, estará en condiciones de transferir sus habilidades a un sistema diferente.

Con el objeto de aclarar las ideas, y empezar a entrar en materia, vamos a mostrarle como escribir el programa de la figura 1.5 en lenguaje ENSAMBLADOR. Por comodidad y facilidad didáctica, vamos a utilizar por ahora instrucciones para una computadora genérica, es decir, una computadora que tenga la mayoría de elementos presentes en las restantes computadoras.



Para facilitar las cosas, vamos a mostrar el proceso gráficamente, **figura 1.7**.

Figura 1.7 *Solución visual del problema del detector de metales*

Paso 1 → Mover el dato desde el detector de metales hacia una posición de la memoria.

Paso 2 → Comparar el dato almacenado en la memoria con el valor 1.

Paso 3 → Si de la comparación resulta que el dato almacenado NO es 1, el detector no ha percibido la presencia de un metal, y debe regresarse al paso 1 para continuar con una nueva lectura.

Paso 4 → En caso contrario, el valor del dato es 1 (presencia de metal) y debe activarse la alarma sonora.

Paso 5 → Terminar el proceso

Figura 1.8 *Solución en palabras del problema del detector de metales*

Ahora que usted ha visto la solución gráfica del problema, podrá entender la conversión a palabras del mismo proceso, **figura 1.8**.

Ya estamos preparados para escribir la solución en lenguaje ENSAMBLADOR genérico. En realidad, todo lo que tenemos que hacer es utilizar palabras más cortas, y establecer convenios que permitan realizar una escritura uniforme.

Cada paso se numera en forma ascendente, y en cada uno de ellos se almacena la correspondiente instrucción, **figura 1.9**. La columna de "Comentario" se utiliza para dar una explicación de cada instrucción y si la instrucción es muy obvia, no se utiliza.

Paso	Instrucción	Comentario
0001	mover	*dato_detector, memoria_interna*
0002	comparar	*memoria_interna, 1*
0003	saltar_si_diferente_a	*paso 0001* diferente_a
0004	activar	*alarma_externa*
0005	parar	

Figura 1.9 *El programa escrito en lenguaje ENSAMBLADOR genérico.*

El lector debe acostumbrarse a traducir mentalmente las instrucciones escritas en lenguaje ENSAMBLADOR. Para ilustrar este proceso, vamos a realizar las correspondientes conversiones de los pasos mostrados en la **figura 1. 9**.

Paso 0001 → Tomar el dato correspondiente al estado del detector y almacenarlo en una posición de memoria en el interior de la computadora.

Paso 0002 → Comparar el dato almacenado en la memoria con el valor 1.

Paso 0003 → Si de la comparación resulta que el dato almacenado es diferente al valor 1, debe saltar al paso 0001 para continuar con la ejecución del programa.

Paso 0004 → En caso de que el dato almacenado sea igual al valor 1, se debe activar la alarma sonora.

Paso 0005 → Detener la ejecución del programa.

El lenguaje ENSAMBLADOR de las computadoras reales es aún más compacto que el mostrado en la figura 9. Una versión más cercana a la realidad se muestra en la **figura 1.10**.

Paso	Instrucción	Comentario
0001	mov	*detector, mem*
0002	comp	*mem, 1*
0003	sdif	*0001*
0004	escr	*alarma, 1*
0005	fin	

Figura 1.10 *El programa escrito en lenguaje ENSAMBLADOR genérico, pero más compacto.*

Simplemente, hemos utilizado palabras más cortas. Vamos a repetir, nuevamente, la traducción mental de instrucciones. Sin embargo, le sugerimos que observe con cuidado la instrucción del paso 0004, ya que en ella hemos realizado una mejora que confiere mayor claridad al proceso de programación.

En lugar de indicar que se activa la alarma sonora, es preferible escribir (enviar) un dato al dispositivo físico: 1 para activar, 0 para apagar.

Más adelante estudiaremos programas similares a los mostrados en las figuras 1.9 y 1.10, pero utilizando el lenguaje ENSAMBLADOR de un microcontrolador real.

El diagrama de bloques o de flujo

Es bastante común, para facilitar su trabajo, que los programadores utilicen elementos gráficos para realzar visualmente las características de los componentes que hacen parte de la solución de un problema. Por este motivo, no es tampoco de extrañar que los programas de las computadoras se diseñen a partir de esquemas concretos,

Paso 0001 → Mover el dato correspondiente al estado del detector de metales, almacenándolo en la posición de memoria denominada *mem*. No debe preocuparse por el nombre utilizado para esta posición de memoria. Es suficiente con saber que todos los elementos internos de una computadora poseen algún nombre en particular.

Paso 0002 → Comparar el dato almacenado en la posición de memoria (denominada *mem*) con el valor 1.

Paso 0003 → Si de la comparación resulta que el dato guardado en la memoria (proveniente del detector) es diferente al número 1, se debe saltar al paso 0001 para continuar con una nueva lectura del estado del detector.

Paso 0004 → En caso de que el dato guardado en *mem* sea igual al valor 1, es decir, se detecta la presencia de un objeto metálico, se debe *escribir* el número 1 en la dirección asignada a la alarma sonora. Esta acción inicia el sonido de la alarma. En caso de necesitar apagarla (quizá un poco más adelante) se utilizaría la instrucción: **escr *alarma*, 0** (escribir el número 0 en la dirección asignada a la alarma).

Paso 0005 → Detener la ejecución del programa.

lo que ha dado origen a una representación denominada DIAGRAMA DE BLOQUES, conocida también con el nombre de DIAGRAMA DE FLUJO. Observe a continuación el programa de la figura 1.9, pero presentado mediante un diagrama de bloques, **figura 1.11**.

En esta construcción se observan varios elementos que es conveniente enumerar. En primer lugar, las figuras con forma rectangular representan acciones concretas que deben ser ejecutadas por la computadora, **figura 1.12**.

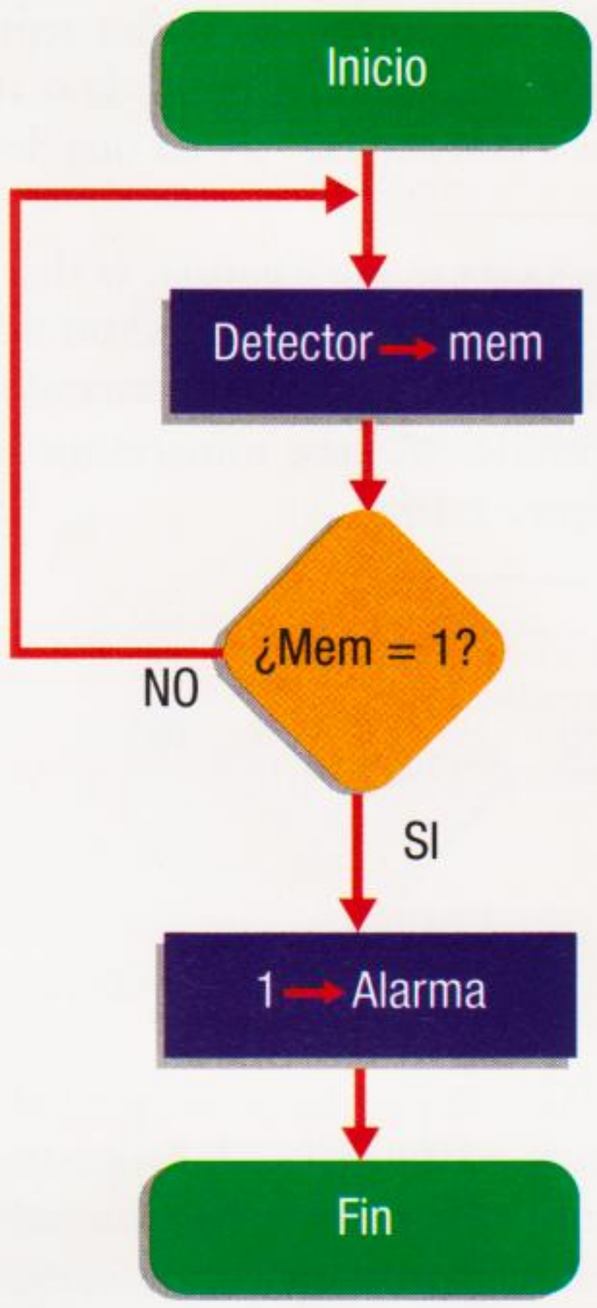


Figura 1.11 *Diagrama de bloques para el programa del detector de metales*

Figura 1.12 *Una acción simple*

Las flechas indican el flujo que debe seguir la computadora para ejecutar las acciones indicadas. No es necesario numerar los elementos gráficos, ya que estas indican con claridad el camino que debe seguirse, **figura 1.13**.

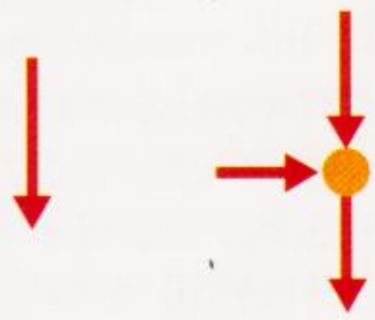


Figura 1.13 *Flechas que indican el flujo de ejecución del programa*

Un figura con forma de rombo representa una condición que debe ser evaluada o analizada, y en base a este análisis, tomar una decisión.

Si la respuesta es satisfactoria, es decir, si se cumple la condición, debe tomarse un determinado camino; en caso contrario, cuando no se cumple la condición, debe tomarse un camino diferente, figura 1.14.

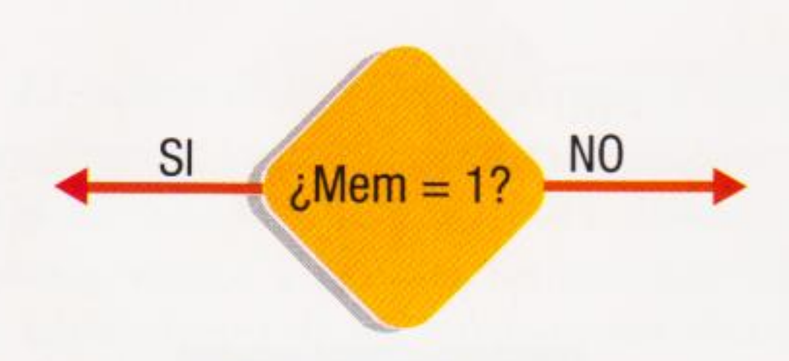


Figura 1.14 *Rombo que indica una decisión*

Un rectángulo con los extremos redondeados indica una acción especial. En particular, si la figura contiene la palabra *Inicio* se indica a la computadora el punto en donde comenzar el programa.

Si el componente contiene la palabra *Fin*, se indica a la computadora que el proceso ha terminado, **figura 1.15**.

Figura 1.15 *Inicio y Fin del programa*

Los textos presentes en los diversos componentes hacen referencia a las acciones que se deben ejecutar. La información que se entrega en los párrafos anteriores brinda una perspectiva de las posibilidades visuales de las que disponemos al elaborar un programa. Naturalmente, en programación existen otros elementos que deben considerarse, pero las estudiaremos cuando sean necesarios.

De la computadora al microcontrolador

En esta primera visión del mundo de la programación, sólo nos resta observar un detalle de gran interés: ¿es lógico utilizar una poderosa y costosa computadora personal para detectar la presencia de objetos metálicos? Es evidente que en el caso de que existan alternativas más simples y económicas, éstas deben ser escogidas. Pero, ¿cuáles son estas alternativas? A la pregunta planteada en el párrafo anterior se puede responder haciendo referencia al componente electrónico más notable de nuestro tiempo: ***el microcontrolador***.

Desde otro punto de vista, podemos definir un microcontrolador como una computadora en miniatura alojada en un solo circuito integrado, pero con la capacidad suficiente para resolver problemas específicos de diferente complejidad, con lo que se obtiene un significativo ahorro en los recursos físicos utilizados, **figura 1.16**.

La arquitectura y el modo de operación del microcontrolador se estudian en la sección de teoría de esta obra. Los aspectos relacionados con las técnicas de programación aplicadas al microcontrolador serán descritos completamente a lo largo de las próximas lecciones.

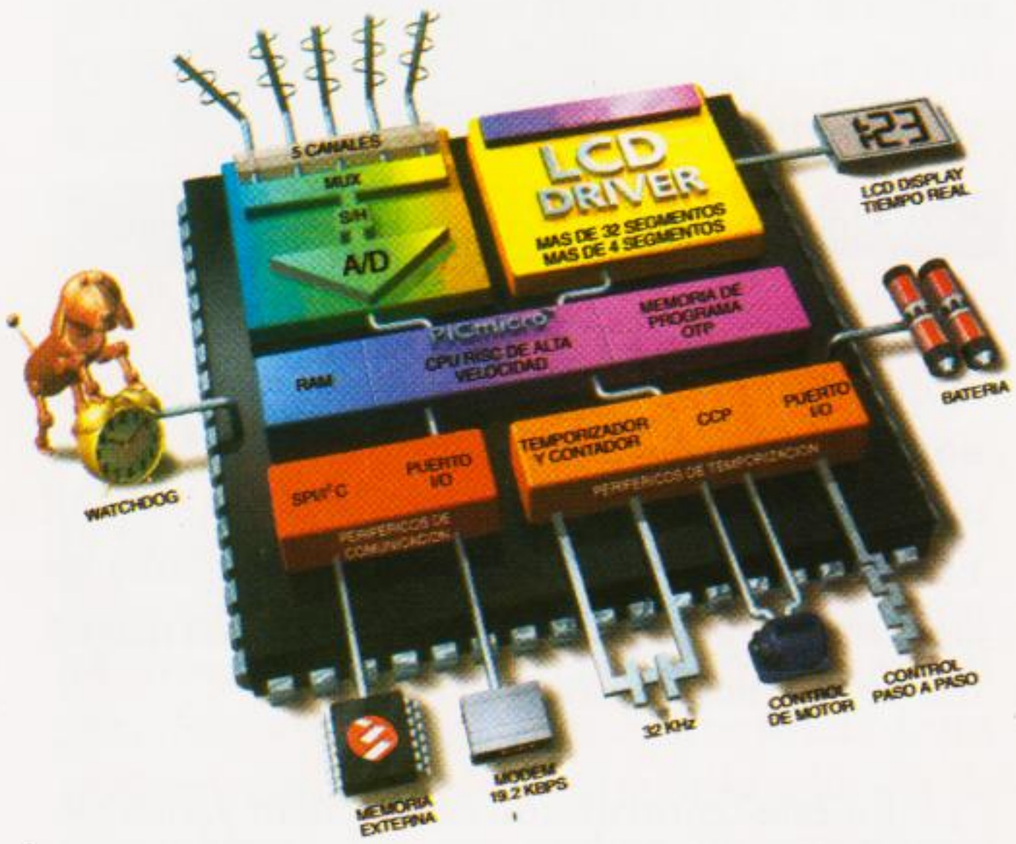
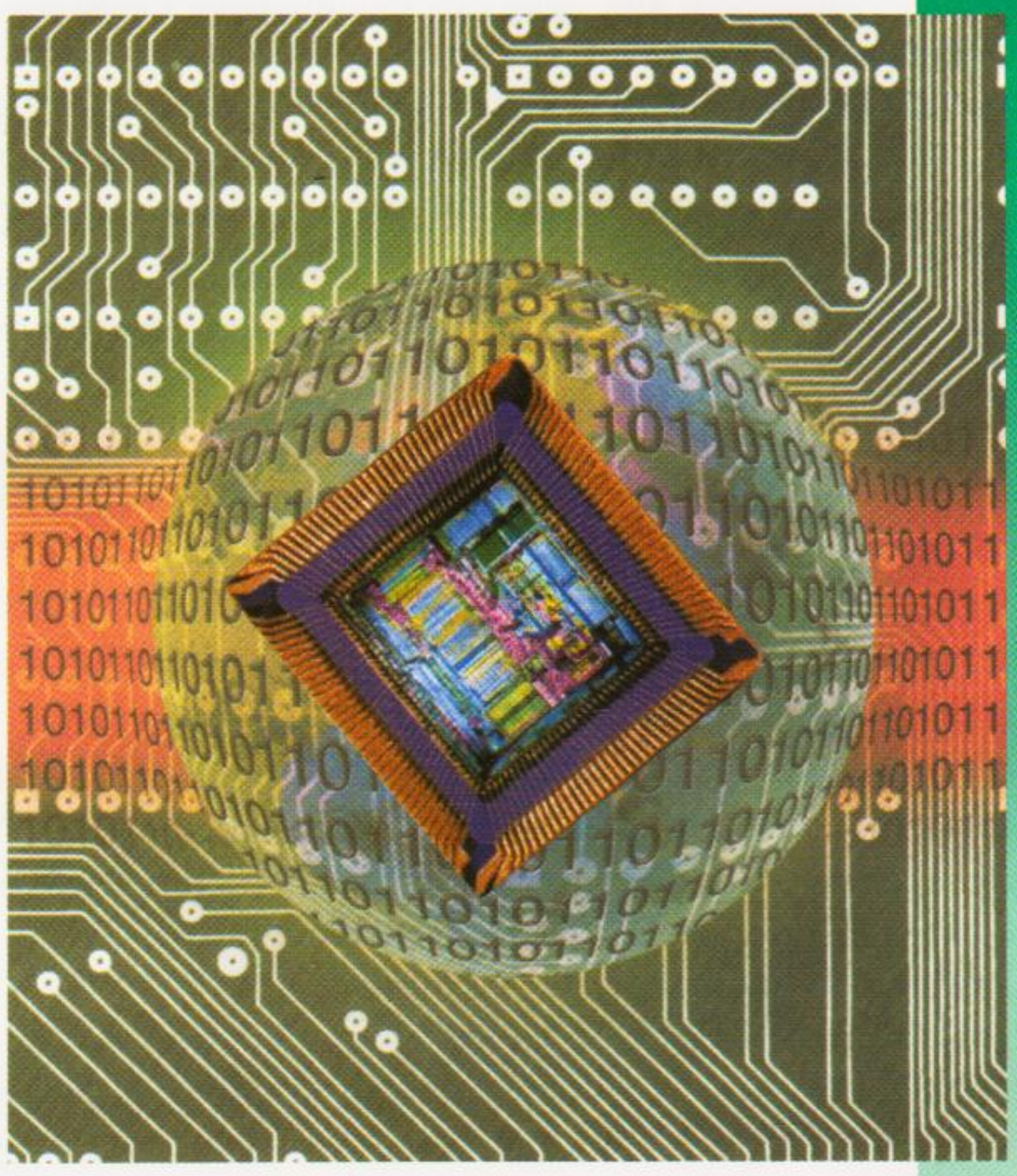


Figura 1.16 *Estructura interna de un microcontrolador*

LECCIÓN 2



SISTEMAS DE NUMERACIÓN EN LA PROGRAMACIÓN DE MICROCONTROLADORES

Los seres humanos trabajamos con el sistema decimal, los microcontroladores trabajan con el sistema binario y para facilitar la escritura de los programas de estos, se trabaja en el sistema hexadecimal. Por esto es importante conocer estos dos últimos sistemas de numeración si queremos aplicar en forma eficiente esta tecnología.

Los microcontroladores son sistemas cuyo comportamiento no se define en el momento de su fabricación. Para que un microcontrolador sea útil, se debe grabar en su memoria interna el conjunto de acciones que debe ejecutar. Una vez hecho esto, se dice que el microcontrolador *ha sido programado,* y se encuentra preparado para ejecutar los procesos de control que le han sido encomendados. Habitualmente, un microcontrolador embebido (*embedded*) o incluido en un circuito de control electrónico, mantiene su programa original, en tanto no se presente ningún desperfecto en el circuito integrado.

¿Qué técnica se utiliza para programar un microcontrolador? En la sección dedicada a los conceptos básicos, se mencionó que las instrucciones del programa se deben codificar utilizando el lenguaje llamado ENSAMBLADOR. Esto, aunque en principio parece correcto, no refleja exactamente lo que pasa en el interior del microcontrolador. En última instancia, los microcontroladores solamente comprenden el lenguaje de los "unos" y los "ceros" o sistema binario; en otras palabras, son circuitos digitales.

El programador, inicialmente, codifica la solución del problema en lenguaje ENSAMBLADOR. A continuación, utilizando una computadora de soporte, transforma el programa convirtiéndolo a código de MÁQUINA (unos y ceros). Este código binario se graba en la memoria del microcontrolador, y es el que realmente se ejecuta al iniciar el ciclo de operación del sistema.

Estructura de los datos

Un concepto fundamental a la hora de programar un microcontrolador, es el relacionado con la manipulación de los datos provenientes del mundo exterior. Suponga, por un momento, que usted decide construir un sistema que mida la temperatura de una habitación y que active una alarma sonora cuando la temperatura sobrepase los 26°C.

¿Cómo hace el microcontrolador para conocer y manipular los datos que corresponden a la temperatura del medio ambiente? Recuerde: todos los datos manejados por un microcontrolador son de tipo *digital.* Esto significa que la información se representa internamente mediante niveles de voltaje discretos. Por ejemplo, un nivel de voltaje de 0 voltios se representa con el número cero (0), mientras que un nivel de voltaje de +5 voltios se representa con el número uno (1), **figura 2.1**.

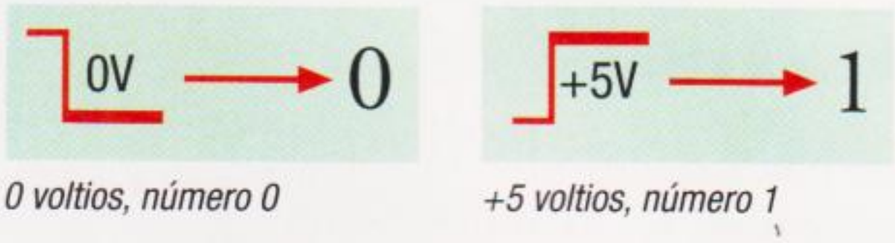


Figura 2.1 *Representación de un dígito binario*

¿Cómo se puede representar una temperatura de 20°C, utilizando para ello unos y ceros? Una solución a este problema se basa en el uso del sistema binario (base dos).

El sistema binario se caracteriza por emplear únicamente los dígitos 0 y 1. Para empezar a entrar en materia, vamos a mostrarle cómo se pueden representar las temperaturas 0°C y 1°C en este sistema. Para el ejemplo planteado, la representación es directa: basta con utilizar un simple dígito binario (también denominado *bit*).

Temperatura	Número binario
0°C	0
1°C	1

Pasemos ahora a representar con números binarios las temperaturas 0°C, 1°C, 2°C y 3°C. Si sólo disponemos de dos valores en el sistema binario, 0 y 1, ¿cómo es posible representar cuatro temperaturas diferentes? La solución se obtiene agregando a la izquierda del número binario un segundo dígito (*bit*) con el cual se puedan crear combinaciones que no se repitan.

Temperatura	Número binario
0°C	0 0
1°C	0 1
2°C	1 0
3°C	1 1

El dígito (*bit*) situado más a la derecha del número binario, toma los valores 0 y 1 para las temperaturas 0°C y 1°C (exactamente igual que antes). Sin embargo, observe que esta situación se repite para las temperaturas correspondientes a 2°C y 3°C.

La clave para diferenciar los dos rangos de temperatura (0°C -1°C y 2°C -3°C) está en el *bit* más a la izquierda del número binario. Este *bit* toma el valor 0 para las dos primeras temperaturas (0°C y 1°C), y adquiere el valor 1 para las dos temperaturas restantes (2°C y 3°C). De este modo, aunque el *bit* de la derecha se repita en los rangos mostrados, la combinación de los dos bits permite representar las cuatro temperaturas de manera única.

Temperatura	Número binario
0°C	0
1°C	1
2°C	0
3°C	1

Representemos ahora las temperaturas 0°C, 1°C, 2°C, 3°C, 4°C, 5°C, 6°C y 7°C. Cómo podrá adivinar el lector, es necesario agregar un nuevo bit en el extremo izquierdo del número binario, de modo que se creen combinaciones únicas. Observe el resultado:

Temperatura	Número binario
0°C	0 0 0
1°C	0 0 1
2°C	0 1 0
3°C	0 1 1
4°C	1 0 0
5°C	1 0 1
6°C	1 1 0
7°C	1 1 1

Las combinaciones de dígitos binarios son únicas; esto garantiza que cada temperatura se representa, también, de manera única. Podríamos continuar haciendo crecer la tabla mostrada, pero es evidente que se debe encontrar un método que permita representar cualquier temperatura, sin importar cual sea su valor.

Para empezar, observe el dígito (*bit*) del extremo izquierdo de los números binarios de la tabla anterior. Este dígito vale 0 para las primeras cuatro temperaturas, mientras que vale 1 para las últimas cuatro. Gracias a la presencia de este tercer *bit*, se puede disponer de ocho temperaturas diferentes.

Como regla general, cada *bit* que se agrega a la izquierda de un número binario, permite duplicar la cantidad de elementos que se podían diferenciar antes de agregarlo.

El esquema de la **figura 2.2** ilustra el peso o valor relativo que tiene el tercer dígito (de derecha a izquierda) en cualquier número binario.

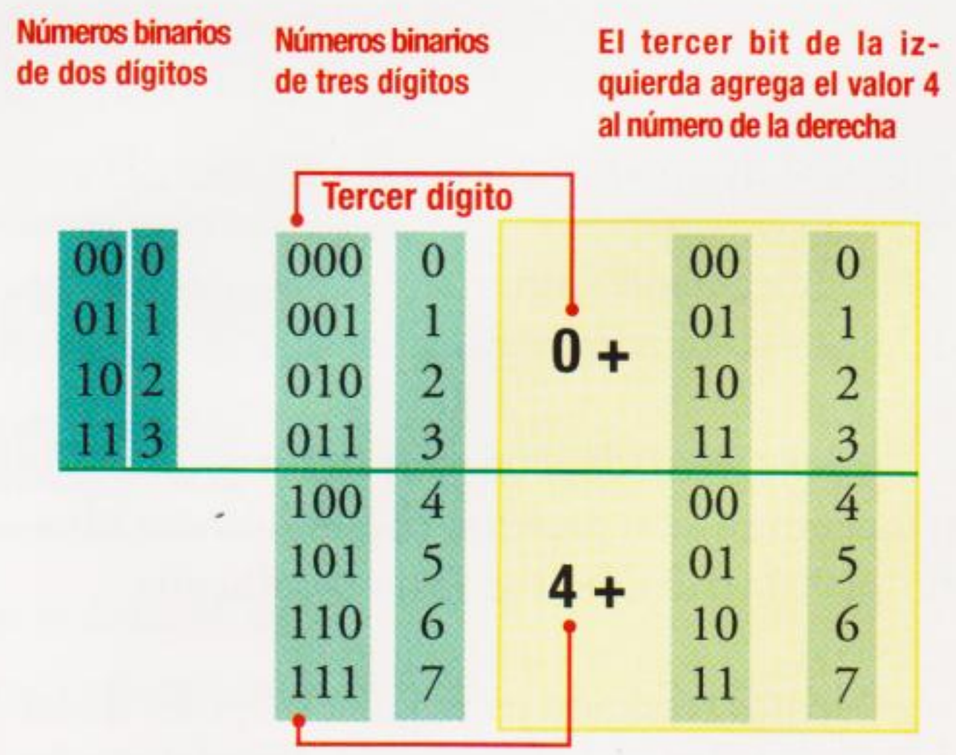
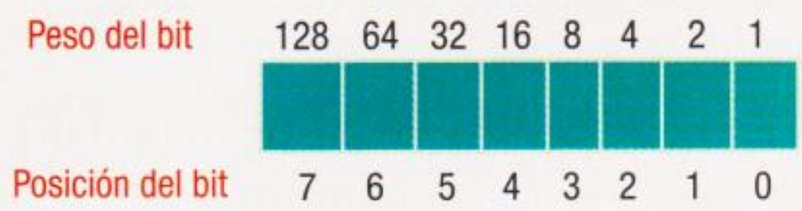


Figura 2.2 *El peso de un bit de acuerdo con su posición*

El efecto de colocar el valor 1 al tercer dígito es similar a sumar 4 al número binario de la derecha. En este sentido, el peso del tercer dígito es 4.

Al hablar de un *bit*, es conveniente conocer su posición dentro del número binario. Es usual

que los dígitos binarios se numeren de derecha a izquierda, en orden creciente, empezando con el valor 0. Para ilustrar los conceptos vistos hasta ahora, presentamos la estructura que tiene un número binario de 8 bits, con los pesos asociados a cada dígito binario, **figura 2.3**.



"El peso de cada *bit* es el doble que el peso del *bit* situado a su derecha"

Figura 2.3 *Pesos asociados a cada bit*

Los pesos permiten establecer el valor decimal de un número binario. Observe el número binario 1011, correspondiente a una temperatura de 11°C. Si queremos encontrar el valor decimal a partir de esta representación binaria, basta simplemente con sumar el peso de los dígitos cuyo valor es 1.

Número binario	1		**0**		1		1		
Pesos	8		4		2		1		
	(8x1) + (4x0) + (2x1) + (1x1) =								
Número decimal	8	+	**0**	+	2	+	1	=	**11**

"Únicamente importan los pesos de los *bits* del número binario, cuyo valor es 1"

Fácil, ¿verdad? ¿Podría el lector indicar qué temperatura se representa con el número binario 11001101? Observe abajo la solución:

Hemos resaltado en negrilla los pesos de los *bits* cuyo valor es 1. El número decimal resulta de la suma de los pesos de los dígitos binarios cuyo valor es 1. El dígito situado más a la izquierda tiene el peso más alto, mientras que el dígito situado más a la derecha es el dígito que tiene el menor peso. El dígito con mayor peso se denomina MSB (*Most Significant Bit*), mientras que el dígito con el menor peso se denomina LSB (*Least Significant Bit*).

Figura 2.4 *Los dígitos MSB y LSB*

Los primeros 16 números binarios son de gran importancia en el dominio de la electrónica digital, de los microcontroladores, de los microprocesadores y de las computadoras. Estos números forman la base del sistema hexadecimal, el cual consiste de 16 símbolos diferentes. En la tabla de la **figura 2.5** se muestran los números del 0 al 15 en su representación binaria, decimal y hexadecimal. Además, se indica cómo derivar los valores decimales a partir de los pesos asociados a los *bits* cuyo valor es 1.

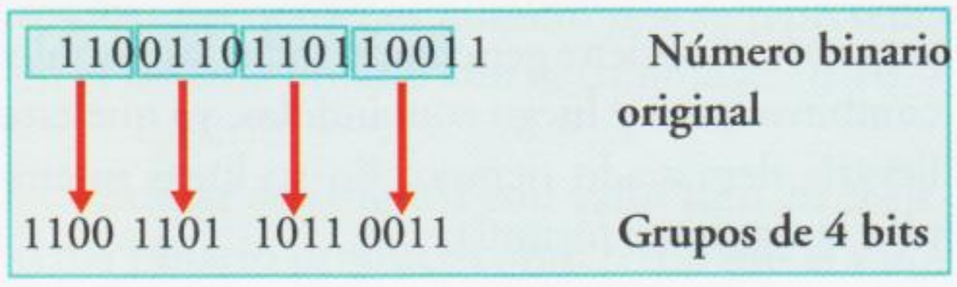
En el sistema hexadecimal, los últimos seis números se representan con las letras A, B, C, D, E y F, tomadas todas ellas del alfabeto. La conversión de un número binario al sistema hexadecimal no cambia el valor del número representado. Sin embargo, es una norma en electrónica utilizar números hexadecimales en lugar de números binarios ¿por qué? Observe la siguiente simplificación: dado el número binario 1100110110110011, encuentre el número hexadecimal equivalente.

Número binario	1		1		0		0		1		1		0		1	
Peso de los dígitos binarios	128		64		32		16		8		4		2		1	
	(128x1)+(64x1)+(32x0)+(16x0)+(8x1)+(4x1)+(2x0)+(1x1)															
Número decimal	128	+	64	+	0	+	0	+	8	+	4	+	0	+	1	= **250**

Número binario	Equivalente Hexadecimal	Equivalente Decimal	Derivación decimal a partir de los dígitos binarios
Peso 8 4 2 1			8 4 2 1
0 0 0 0	0	0	0+0+0+0
0 0 0 1	1	1	0+0+0+1
0 0 1 0	2	2	0+0+2+0
0 0 1 1	3	3	0+0+2+1
0 1 0 0	4	4	0+4+0+0
0 1 0 1	5	5	0+4+0+1
0 1 1 0	6	6	0+4+2+0
0 1 1 1	7	7	0+4+2+1
1 0 0 0	8	8	8+0+0+0
1 0 0 1	9	9	8+0+0+1
1 0 1 0	A	10	8+0+2+0
1 0 1 1	B	11	8+0+2+1
1 1 0 0	C	12	8+4+0+0
1 1 0 1	D	13	8+4+0+1
1 1 1 0	E	14	8+4+2+0
1 1 1 1	F	15	8+4+2+1

Figura 2.5 Equivalente binario, hexadecimal y decimal

El primer paso que se debe dar para resolver el problema propuesto consiste en dividir el número binario en grupos de 4 bits: se empieza agrupando los 4 bits a la derecha del número binario, se agrupan luego los 4 bits ubicados inmediatamente a la izquierda, y el proceso continúa hasta que se agotan todos los dígitos del número binario original. Si el último grupo creado no contiene exactamente 4 bits, se recurre a completarlo agregando ceros (no significativos) a su izquierda. Apliquemos este procedimiento al número binario 1100110110110011.



El segundo paso en la solución del problema consiste en buscar en la tabla de números hexadecimales el grupo binario a ser reemplazado, escribiendo en su lugar el número hexadecimal correspondiente, **figura 2.6** .

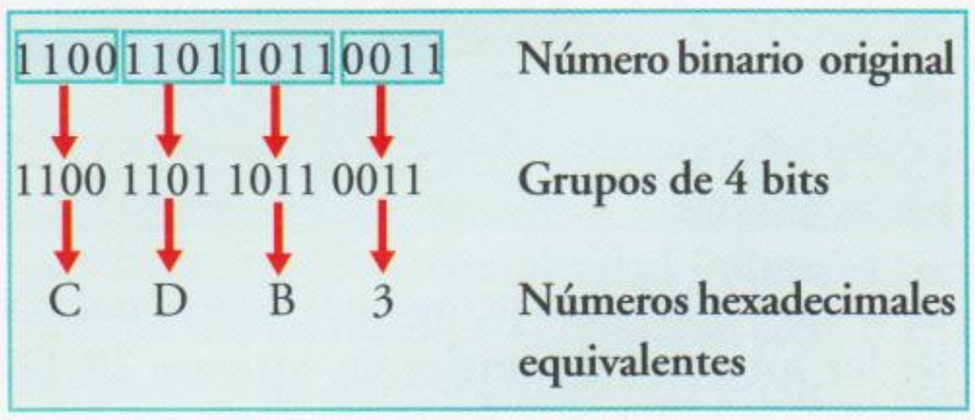


Figura 2.6 Generación de un número hexadecimal

Una consecuencia positiva de la representación hexadecimal estriba en que es mucho más fácil recordar el número CDB3 en base 16, que su equiva-

lente binario 1100110110110011. Otra ventaja que se deriva del uso de los números hexadecimales tiene que ver con la disminución del espacio empleado para su representación en documentos escritos.

En los próximos apartados vamos a establecer algunas definiciones que serán útiles más adelante.

Bit

Un *bit* es la unidad mínima de información en un sistema binario, y puede tomar sólo uno de estos valores: 1 ó 0.

Nibble

Un *nibble* es un grupo de 4 *bits*. El *nibble* es útil para representar conjuntos de dígitos decimales, como se verá a continuación.

BCD

Un dígito decimal: 0, 1, 2, 3, 4, 5, 6, 7, 8 o 9 se puede representar mediante 4 dígitos binarios (un *nibble*). La siguiente tabla muestra los dígitos decimales y su equivalente binario, al cual se lo conoce con el nombre BCD (*Binary Coded Decimal*).

Dígito decimal	Dígito BCD
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

Algunos sistemas permiten la representación de los números decimales en formato BCD. Cuando se utiliza esta forma de representación, se reemplaza cada dígito decimal por su equivalente en BCD, uno después de otro. El resultado que arroja este proceso difiere del que se obtendría aplicando la numeración binaria basada en pesos. Para ilustrar lo anterior, consideremos el número 32; su representación en binario es: 00100000. En la **figura 2.7** puede apreciar su representación en BCD.

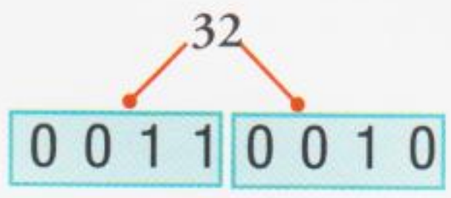


Figura 2.7. *Representación en BCD del número 32*

Byte

Un *byte* es un grupo de 8 bits. En un *byte*, el *bit* de mayor peso, usualmente el de la izquierda, se denomina *bit* MSB; el *bit* de menor peso, usualmente el de la derecha, se denomina *bit* LSB. El *byte* es una de las estructuras de datos más utilizadas en todos los sistemas de computación.

Word

Una **palabra** (***word***) es un conjunto de 16 *bits*, y equivale a la unión de 2 *bytes*. Ocasionalmente, es necesario agrupar un conjunto de 32 bits como si se tratara de una sola entidad. Esta nueva estructura (4 *bytes*) se conoce con el nombre de palabra doble (*double*).

La **figura 2.8** presenta un resumen de los elementos vistos.

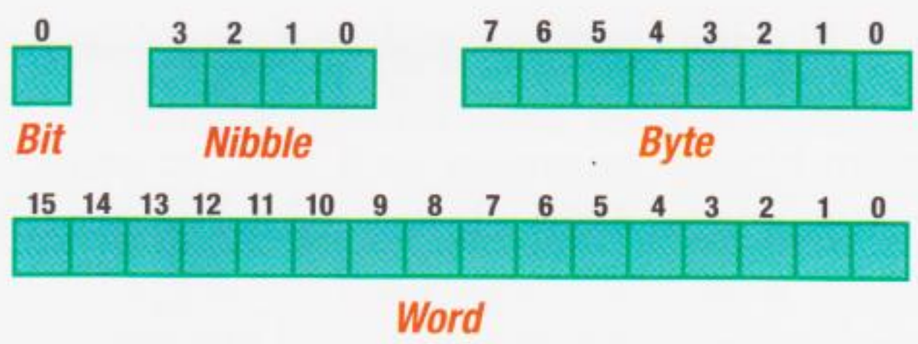


Figura 2.8 *Representaciones binarias*

Rango de los números binarios

Si utilizamos únicamente 2 dígitos binarios, podremos diferenciar entre 4 entidades distintas: 00, 01, 10 y 11. ¿Cuántos elementos podremos diferenciar con 16 bits? La respuesta a esta pregunta no se obtiene generando todas las posibles combinaciones y luego contándolas, ya que esto llevaría demasiado tiempo. En su lugar se emplea la siguiente fórmula:

Número de combinaciones = 2^n

en donde n es el número de dígitos binarios. Aplicando la fórmula al problema planteado se obtiene:

Número de combinaciones = 2^{16} = 65536

Operaciones básicas

Los microcontroladores permiten el uso extensivo de las operaciones matemáticas básicas: es posible sumar o restar números binarios, realizar operaciones de tipo lógico y desplazar el contenido binario de un registro o posición de memoria. A continuación, vamos a ilustrar las operaciones de suma y resta binaria.

La suma de dígitos binarios

La suma de números binarios es similar a la suma de números decimales. Basta con recordar que en el caso del sistema binario, sólo se utilizan dos dígitos: el cero y el uno. La **figura 2.9** permite entender el mecanismo empleado cuando se suman números binarios.

0	0	1	1
+ 0	+ 1	+ 0	+ 1
0	1	1	10

Figura 2.9 *La suma binaria*

La primera suma es evidente: al sumar dos ceros se obtiene un cero. La segunda suma, al igual que la tercera, tampoco presenta ninguna dificultad: al sumar el número uno con el número cero se obtiene como resultado el número uno.

El caso más interesante corresponde a la cuarta columna: si suma el número uno con el número uno, obtiene como respuesta el número dos. Recuerde: el número dos se representa en el sistema binario con la combinación 10.

Sumar dos dígitos con valor uno en el sistema binario produce como resultado el valor cero. El uno que sobra se acumula y pasa a la siguiente columna.

La operación binaria: 1 + 1 + 1, se puede resolver recurriendo a sumas sucesivas. Observe la **figura 2.10**. En el paso (a) se suman los dos primeros unos. La respuesta es cero y "llevo" uno. En el paso (b) se toma el valor calculado en el paso anterior (10) y se agrega al tercer uno que teníamos pendiente por sumar. La respuesta final es 11.

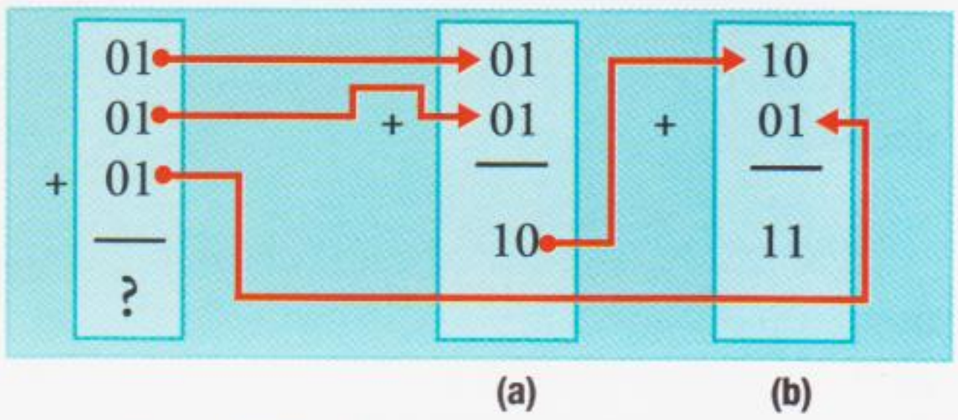


Figura 2.10 *Suma de tres dígitos binarios cuyo valor es 1*

Suma de números binarios

La suma de números binarios es similar a la suma de números decimales: se empieza con los dígitos situados a la derecha del número binario, sumándolos entre sí, y calculando, tanto el resultado de la suma como el valor del acarreo (*carry*) que se transfiere a la siguiente columna.

El proceso se repite para cada nueva columna a la izquierda, pero tomando en cuenta el *bit* de acarreo que viene de la columna más a la derecha.

El proceso descrito en el párrafo anterior puede resultar difícil para quien no tenga experiencia previa con los sistemas binarios. Por esta razón, sugerimos que las sumas binarias que tengan algún grado de complejidad (números con muchos dígitos binarios) se resuelvan convirtiendo primero los números al sistema decimal, realizando a continuación la suma y por último, pasando la respuesta a su equivalente binario (un método más simple consiste en utilizar una calculadora que tenga incluida la posibilidad de sumar números binarios).

Números binarios negativos

En apartados anteriores hemos visto como representar números decimales en notación binaria. Pero podríamos preguntar: ¿y si el número decimal es negativo? La respuesta a esta cuestión es clave para las operaciones de resta binaria. En primer lugar, antes de entrar en materia, vamos a recordar como se representa (utilizando ocho bits) el número decimal 3:

Decimal	Binario
3	0 0 0 0 0 0 1 1

¿Cómo se representa, en binario, el número decimal –3 ?.

Para empezar, si usted suma el valor 3 al número –3, debe obtener como respuesta el número 0. Entonces, ¿qué valor debe sumarse al número binario 00000011 (3 decimal) para obtener el número 00000000 (0 decimal)? Observe la respuesta en la **figura 2.11**.

Decimal	Binario
3	0 0 0 0 0 0 1 1
+ -3	+ 1 1 1 1 1 1 0 1 ←
0	1 0 0 0 0 0 0 0 0

Figura 2.11 *Número binario negativo*

Supongamos, por un momento, que el *bit* de acarreo no importa y puede desecharse. Si este fuera el caso, podríamos asumir que el valor 11111101 equivale al número –3, ya que al sumarlo al número 3 produce el valor 0.

¿Cómo puede representarse, en general, un número decimal negativo usando el sistema binario? La respuesta está contenida en el algoritmo denominado ***complemento a dos*** lo cual se logra de la siguiente manera:

1. Se convierte el número decimal a su representación en binario.
2. Se invierten los dígitos binarios: aquellos cuyo valor es uno se cambia por cero, y aquellos cuyo valor es cero se cambian por uno. A esto se lo llama **complemento a uno.**
3. Por último, se suma el *bit* 1 al número obtenido en el paso anterior. El resultado es el número negativo buscado.

Ejemplo:

Representar en binario el número –25.

Aplicación del algoritmo

1. Convertir el número 25 a binario.

Peso	128	64	32	16	8	4	2	1
Número 25	0	0	0	1	1	0	0	1

2. Complementar los dígitos binarios.

Número 25	0 0 0 1 1 0 0 1
Complemento a uno	1 1 1 0 0 1 1 0

3. Sumar al número en complemento a uno con el valor 1.

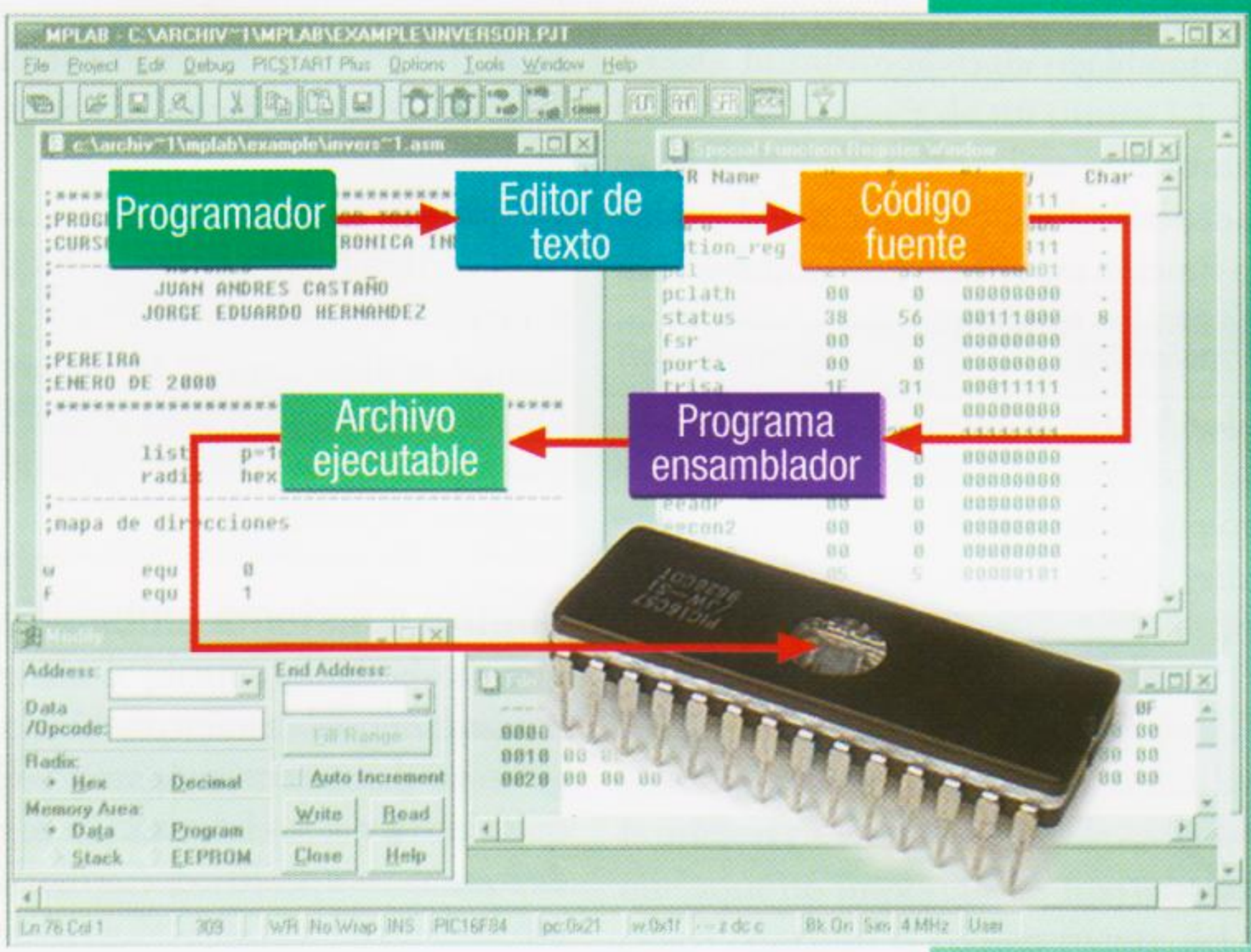
Complemento a uno	1 1 1 0 0 1 1 0	
Suma con bit 1	0 0 0 0 0 0 0 1	
Número (-25)	1 1 1 0 0 1 1 1	**Respuesta**

Resta de números binarios

Del apartado anterior debe resultar evidente que el proceso para realizar restas binarias consiste simplemente en encontrar el número negativo del valor a restar, sumándolo a continuación al número positivo. Esto es equivalente (en decimal) a una operación del tipo: 48 –32 = 48 + (-32).

De nuevo, al igual que en la suma binaria, es más cómodo realizar primero la conversión de los números binarios a su equivalente decimal antes de proceder a realizar la resta. El resultado obtenido se convierte al equivalente binario negativo, obteniéndose la respuesta deseada. En capítulos posteriores tendremos la ocasión de aplicar este método.

LECCIÓN 3



PROGRAMACIÓN DE MICROCONTROLADORES

Uno de los aspectos más importantes en el diseño y en la ejecución de proyectos y aparatos con microcontroladores, es la planeación y escritura adecuadas de sus programas, los cuales varían según la marca y el modelo del dispositivo utilizado, los circuitos externos a él conectados y la función a realizar. A partir de esta lección iremos explicando, paso a paso, la metodología y las técnicas apropiadas para hacerlo.

La solución de un problema de control electrónico, basado en microcontroladores, incluye dos etapas fundamentales: escribir el programa en lenguaje ensamblador y generar el archivo binario ejecutable que debe grabarse en la memoria del microcontrolador. Para empezar, estudiaremos el proceso de ensamblado de código.

El ensamblador está conformado por varios módulos independientes, cada uno de los cuales cumple una función específica. Los módulos más importantes son los siguientes:

Ensamblador básico. Genera, a partir del código fuente, un archivo binario relocalizable. Este archivo puede ser almacenado en cualquier segmento disponible en la memoria del microcontrolador.

Enlazador (*linker*). Crea, a partir del archivo binario relocalizable, un archivo binario ejecutable. Este código es el que ejecuta directamente el microcontrolador.

Control de librerías (*lib*). Este módulo permite crear archivos binarios que pueden ser unidos (enlazados) con otros bloques de código binario, lo que facilita la reutilización de partes de programas generados en otros proyectos. El uso de librerías simplifica el desarrollo de programas de gran tamaño y complejidad.

La **figura 3.1** muestra la estructura del ensamblador, así como el flujo de información entre sus módulos.

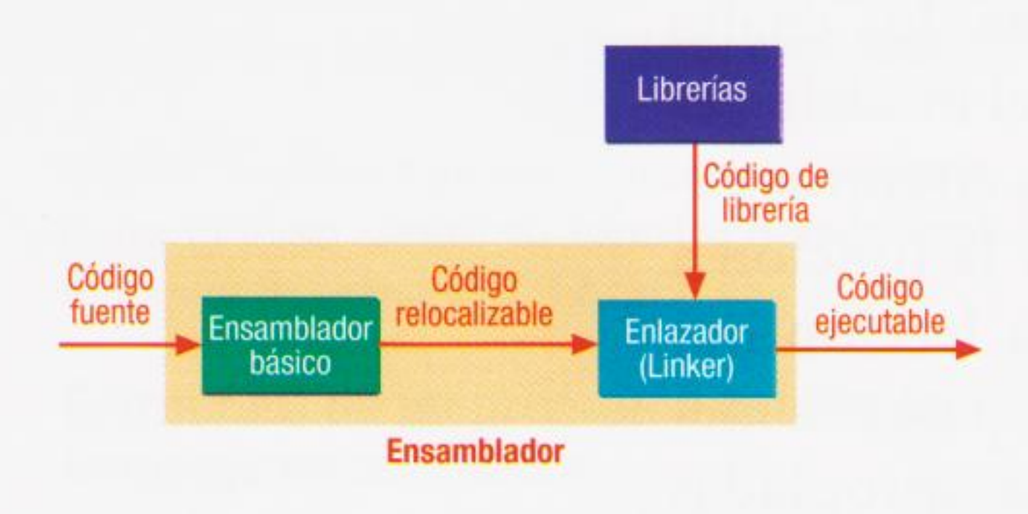


Figura 3.1 *Estructura del ensamblador*

En lo sucesivo emplearemos el término ensamblador para referirnos al conjunto completo de programas que genera un archivo binario a partir del código fuente de un programa de usuario.

Un **programa de usuario** es un programa escrito usando el lenguaje ensamblador. Su objetivo es la solución de un problema de control electrónico.

Un **programa ensamblador** es aquel que recibe el código fuente de un programa de usuario y genera un archivo binario ejecutable. Este código se almacena en la memoria de programa, y es ejecutado por la unidad central de proceso cada vez que el microcontrolador se pone en marcha.

Control o sistema embebido

Un proyecto de control o sistema embebido (*embedded system*) utiliza un microcontrolador para gestionar de manera autónoma los procesos de adquisición de datos y la activación de las líneas de salida conectadas a periféricos externos. Debido a que el precio de un microcontrolador es insignificante comparado con el de una computadora personal (PC) y, además, su tamaño es bastante reducido, se entiende que se lo prefiera en el control de procesos electrónicos y en el diseño de aparatos electrónicos modernos dotados con cierta "inteligencia".

El desarrollo de un proyecto embebido está sujeto a un ciclo de vida particular, según se aprecia en la **figura 3.2**.

Figura 3.2 *Ciclo de vida de un proyecto embebido*

Este ciclo de vida se atiene a principios sólidos utilizados durante décadas por los ingenieros especializados en el desarrollo de programas de tipo general. Un esquema más simple, y tal vez más cercano a la realidad de quien programa un microcontrolador, se muestra en la **figura 3.3**, en donde se indica de manera concreta cuales son los pasos que debe seguir un programador para obtener el programa o código final.

Figura 3.3 *Etapas en el desarrollo de una aplicación para microcontroladores*

En la **figura 3.4** se plantea el método de trabajo que recoge las ideas presentadas previamente y que son la clave para desarrollar un programa con éxito. Aunque el proceso es de naturaleza secuencial, es común que se elabore uno en espiral en el que, cada cierto tiempo, se practique una revisión minuciosa de los objetivos alcanzados hasta el momento. Esto implica una revisión permanente del trabajo desarrollado.

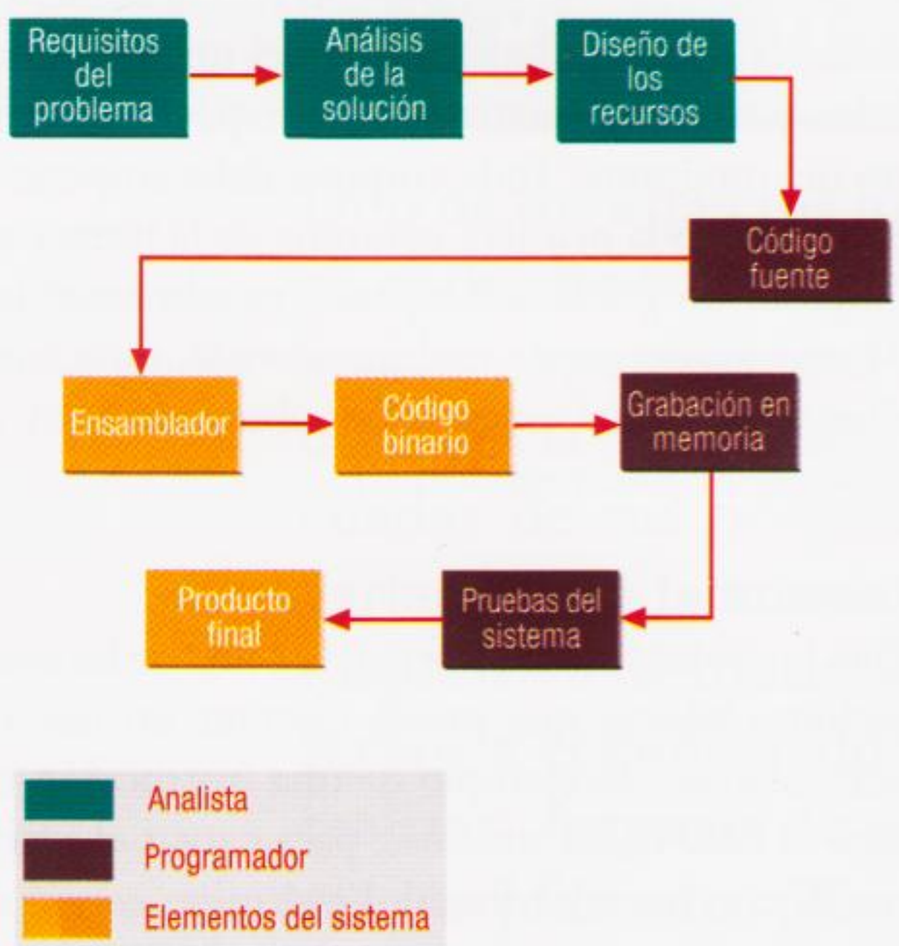


Figura 3.4 *Método de trabajo para el desarrollo de programas basados en microcontrolador*

Archivos generados por el programa ensamblador

Además del código binario, el ensamblador genera un conjunto de archivos adicionales, gracias a los cuales es posible controlar la evolución del proyecto. La extensión que acompaña a cada nombre de archivo, y que consiste en tres letras después de un punto (.), indica cual es el tipo de información que contiene el archivo. Un ejemplo tomado de un ensamblador típico es el siguiente:

Tipo de archivo	Extensión	Ejemplo
Código fuente del programa	ASM	primero.asm
Código binario ejecutable	HEX	primero.hex
Listado del programa	LST	primero.lst
Lista de errores	ERR	primero.err
Código objeto ejecutable	OBJ	primero.obj
Archivo de librería	LIB	rutinas.lib

Un archivo con extensión ASM (ó asm, dado que aquí no importan las mayúsculas o minúsculas) contiene el código fuente del programa. El nombre del archivo (la palabra **primero**, en el ejemplo mostrado) está separado de su extensión mediante un punto, y debe cumplir con las restricciones impuestas por el sistema operativo. El método más simple para crear un nombre correcto consiste en empezar con una letra (y en algunos casos con un carácter de subrayado (_)), agregándole a continuación una combinación arbitraria de letras, números y caracteres de subrayado. Algunos sistemas operativos restringen la longitud del nombre a un máximo de ocho caracteres. Ejemplos válidos de nombres de archivos fuente son los siguientes:

primero.asm
_control.asm
alarma_1.asm
tempX20.asm

Un archivo con extensión **HEX** u **OBJ** contiene el código binario ejecutable, y se deriva automáticamente de un archivo con extensión **ASM**.

El proceso de ensamblado puede generar también, de manera automática, dos archivos de gran importancia práctica. El primero de ellos (con extensión **LST**) es un listado que contiene el texto del programa fuente debidamente formateado, lo que permite documentar con claridad el desarrollo del mismo. Este listado incluye la numeración de páginas, la fecha y la hora en que se hizo el proceso de ensamblado, la tabla de símbolos del programa fuente, la distribución de los datos en la memoria, así como los errores y advertencias que se derivan directamente del código fuente. Las instrucciones se organizan en columnas de acuerdo con una tabulación estándar.

El segundo tipo de archivo (con extensión **ERR**) le sirve al programador para estudiar los errores cometidos (cuando éstos se producen), facilitando el proceso de corrección y depuración del código fuente.

Los tipos de archivo descritos anteriormente pueden diferir de acuerdo con el microcontrolador utilizado.

Términos usuales

Es importante establecer el significado de algunos términos que serán muy utilizados en las próximas lecciones:

Registro. Es un depósito de números binarios que permite guardar datos en forma temporal. El nombre del registro sirve para identificarlo plenamente y establece el tipo de información que almacena. Por ejemplo, el registro **W** (disponible en los microcontroladores de la familia **PIC**) se utiliza en la mayoría de operaciones que manipulan datos. Su nombre viene de la palabra inglesa *work* (trabajo).

Literal. Se refiere a un valor constante, usualmente un número escrito en formato hexadecimal. Ejemplos de literales: 3, 12, 3F.

Estructura de un programa escrito en lenguaje ensamblador

Todos los microcontroladores disponen, como mínimo, de un ensamblador. Ahora bien, debido a su naturaleza particular, cada ensamblador tiene sus propias reglas que deben ser respetadas. Afortunadamente, existen ciertas normas de tipo general, comunes a la mayoría de los ensambladores, que es necesario conocer para reducir el tiempo invertido en el proceso de escribir un programa de usuario. A continuación, comentamos algunas de estas características.

Un programa escrito usando instrucciones del lenguaje ensamblador debe organizarse según un diseño basado en columnas. En la **figura 3.5** se muestra la estructura de un programa escrito en ensamblador.

Etiquetas	Instrucciones	Operandos	Comentarios
inicio	org	0	;Comienzo del programa

Figura 3.5 *División en columnas de un programa fuente escrito en ensamblador*

Cada una de las columnas cumple una función específica, según se indica a continuación.

Etiqueta

Una etiqueta es un nombre con el cual se identifica una posición de memoria del microcontrolador, y sirve para marcar puntos específicos dentro del programa. Toda etiqueta debe empezar a escribirse en la primera columna de la línea y su longitud no puede sobrepasar (usualmente) los 31 caracteres; puede incluir, además, caracteres alfanuméricos, el carácter de subrayado (_) y el símbolo de interrogación (?).

Instrucción

Una instrucción hace referencia a una de las operaciones básicas que puede ejecutar un microcontrolador. Un ejemplo de una instrucción típica es **MOVLW**, utilizada para cargar el registro **W** con un valor literal. El término ***cargar*** se interpreta en este contexto como ***almacenar en***, por lo que la instrucción **MOVLW** puede leerse así: "almacenar un valor literal en el registro **W**".

Operando

Un operando es un elemento utilizado por una instrucción. Algunas instrucciones, debido a su simplicidad, no utilizan operando. Otras, un poco más complejas, requieren el uso de un sólo operando. Por último, existen instrucciones que requieren el uso de dos operandos para cumplir cabalmente su función. El primer operando, cuando está definido, se denomina **operando fuente**. El segundo operando, complemento del anterior, recibe el nombre de **operando destino**. La información fluye desde el operando fuente hacia el operando destino.

Comentario

Un comentario es un bloque de texto que le sirve al programador para documentar sus programas. Los comentarios son ignorados por el ensamblador, y se reconocen porque empiezan con el carácter punto y coma (;).

Tipos de línea que se pueden incluir en un programa

Los programas en ensamblador se deben escribir según un esquema que permita estructurar de manera sólida al trabajo desarrollado. En la **figura 3.6** se muestra la arquitectura general de un programa para microcontrolador.

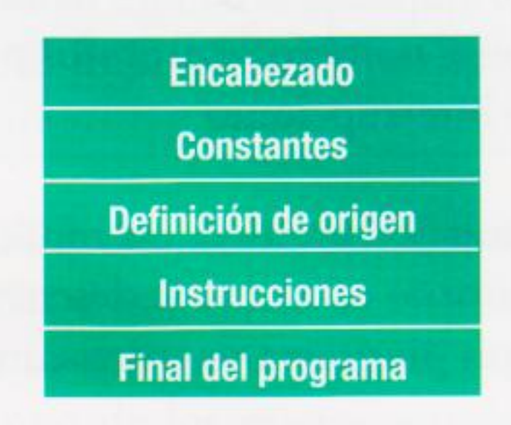


Figura 3.6 *Arquitectura general de un programa en ensamblador*

A continuación presentamos el significado de cada uno de los bloques mencionados.

Encabezado

Es el primer componente del programa, y en él se definen algunas directrices de tipo general, conocidas como directivas, que modifican el funcionamiento del ensamblador. Para ilustrar lo anterior, considere la familia de microcontroladores **PIC**, la cual utiliza el ensamblador **MPASM**. En este ensamblador la directiva **list** indica la versión del microcontrolador utilizado, lo que establece el tipo de programa binario generado. Esta directiva es obligatoria, ya que cada microcontrolador genera su propio código binario y el ensamblador debe adaptarse al modelo utilizado.

Para ilustrar lo expuesto en el párrafo anterior, supongamos que usted utiliza en un proyecto el microcontrolador **16C84**. La directiva que debe incluir al comienzo del programa es la siguiente:

```
list  p=16C84  ; microcontrolador 16C84
```

Constantes

En esta sección se incluyen las definiciones de constantes que son reconocidas en los demás puntos del programa. El uso de constantes simplifica la lectura de un programa, ya que, en lugar de referirse a un valor hexadecimal en particular, se puede utilizar el nombre de la constante asociada. La mayoría de los ensambladores utilizan la palabra **equ** para definir una constante. Por ejemplo, la línea

```
puertoA     equ   05
```

le indica al ensamblador que se puede utilizar la palabra **puertoA** en lugar del número 05.

Definición del origen del programa

Al programar un microcontrolador es necesario indicar explícitamente en que sitio de la memoria se debe almacenar el código binario del programa que está a punto de ser ensamblado. Esto se consigue con la palabra **org** (origen). Por ejemplo, la siguiente línea

```
org          0
```

indica al ensamblador que debe almacenarse, a partir de la línea 0 en la memoria de programa,

el código binario generado. Existen algunas variaciones que indicaremos oportunamente cuando sea necesario.

Instrucciones del programa

Esta sección se utiliza para almacenar las instrucciones del programa. Una instrucción típica incluye los siguientes campos:

etiqueta	código de instrucción	operando(s)	; comentario

A excepción del campo **código de instrucción**, los restantes campos pueden aparecer (o no aparecer) dentro de una línea de programa.

Final del programa

Esta sección contiene una instrucción simple que marca el final del programa. El ensamblador **MPASM** (de la familia de microcontroladores **PIC**) utiliza la siguiente instrucción para conseguir este propósito:

```
end
```

El entorno de programación

El objetivo de cualquier programa de usuario es, en principio, muy simple: leer datos de periféricos externos, calcular, y activar líneas de salida que produzcan cambios en el ambiente externo.

Las siguientes afirmaciones permiten entender la estructura del entorno de programación del microcontrolador:

1. El microcontrolador accede al contenido o a las señales de los componentes externos, a través del uso de **puertos de entrada**.
2. El microcontrolador modifica el contenido o envía señales a los componentes externos, mediante el uso de **puertos de salida**.
3. Los datos leídos en un puerto de entrada son almacenados, en primer lugar, en un **registro interno** del microcontrolador.
4. Cuando un microcontrolador lee muchos datos provenientes de componentes externos (mediante el uso de puertos de entrada), debe almacenar esta información en la **memoria de datos**; esto permite que los datos estén disponibles para ser utilizados más adelante.
5. El microcontrolador posee una **unidad aritmética y lógica** que se encarga de realizar operaciones sobre los datos. De acuerdo con los resultados obtenidos, el microcontrolador toma decisiones sobre la activación de líneas en los puertos de salida.
6. Un sistema embebido (basado en microcontrolador), cumple con el objetivo de control para el cual ha sido diseñado gracias a la existencia de un **conjunto de instrucciones** (**programa**) que indica de manera precisa lo que debe hacerse en cada instante del tiempo.
7. Un programa (conjunto de instrucciones) se almacena en la **memoria de programa** del microcontrolador.
8. Un microcontrolador puede recibir señales del exterior (en instantes del tiempo que no es posible predecir), ante las cuales debe reaccionar sin pérdida de tiempo. Esto implica el abandono temporal de aquello que se está ejecutando, para dar cumplimiento a una función que atienda la **interrupción**. A los procesos implicados en este tema se los conoce con el nombre de **procesos para el manejo de interrupciones**.

La **figura 3.7** presenta un modelo simplificado del entorno en el que se desenvuelve la ejecución de un programa de microcontrolador.

El contador de programa

El microcontrolador utiliza un registro especial, denominado **contador de programa**, para saber cuál es la siguiente instrucción que debe ejecutar. El proceso es simple: al comienzo de la ejecución, el contador de programa apunta a la primera instrucción del programa; cada vez que ejecuta una instrucción, el contador de programa incrementa su valor, de modo que la instrucción que se ejecuta la próxima vez es la siguiente en la memoria de programa.

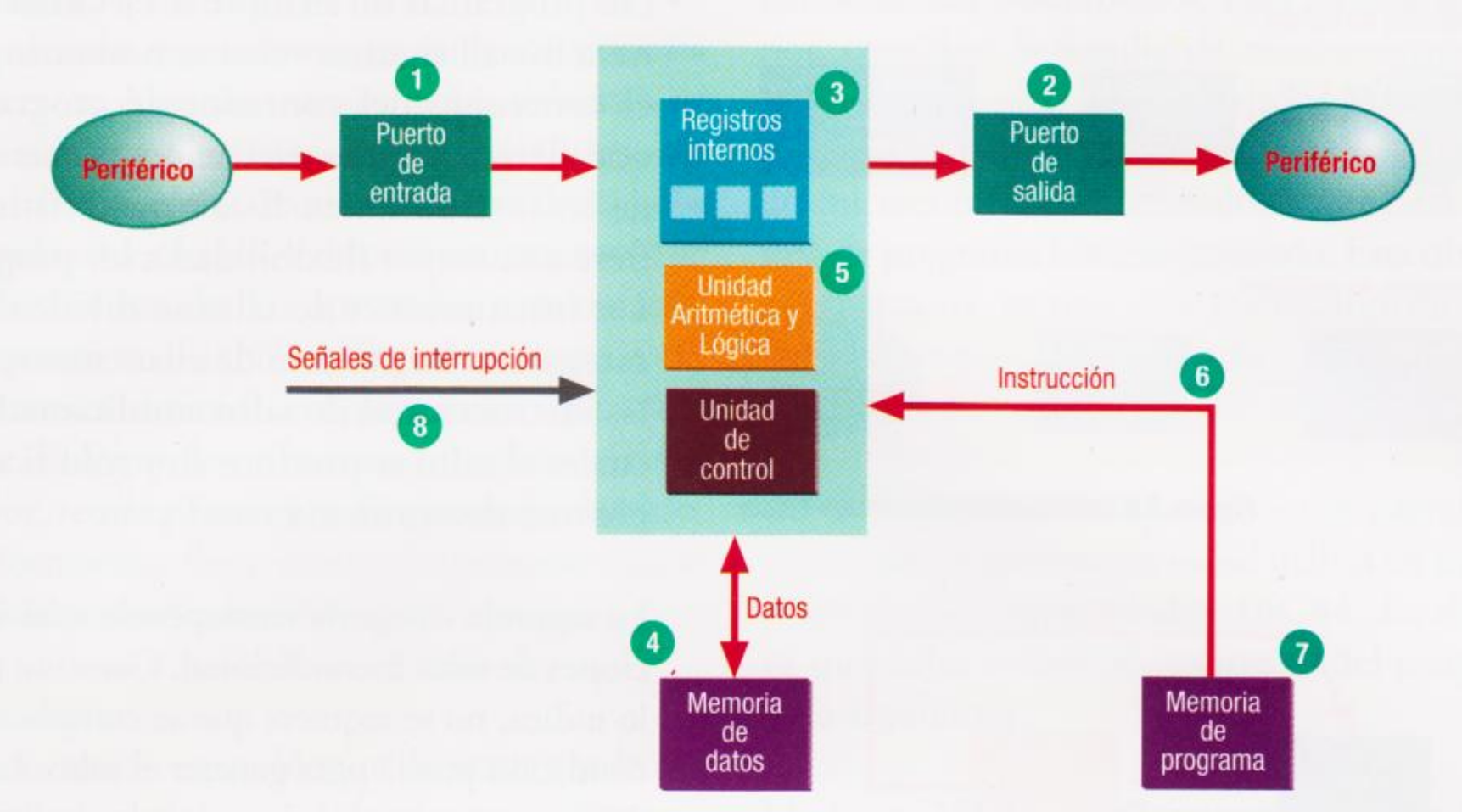


Figura 3.7 El entorno de programación fundamental

La ejecución de las instrucciones, y el cambio en el contador de programa, es un proceso automático y transparente para el programador, lo que significa que se produce bajo el control interno del microcontrolador.

Algunas veces el contador de programa cambia su contenido por un valor que no corresponde a la siguiente posición de memoria. Esto se verá en el siguiente apartado. La **figura 3.8** muestra el flujo probable en la ejecución de un programa.

Conjunto (*set*) de instrucciones

Antes de iniciar el estudio de las técnicas de programación, y su aplicación a problemas concretos en el campo de los microcontroladores, es importante comprender la naturaleza de las instrucciones disponibles y la forma como se organizan jerárquicamente, según las funciones para las que han sido diseñadas.

- Cada instrucción permite actuar sobre un determinado elemento del entorno de programación: algunas de ellas permiten **leer información** proveniente de periféricos externos (mediante puertos de entrada), mientras que otras alteran componentes conectados a los puertos de salida, lo que en cierto sentido implica un proceso de **escribir información**. Este proceso se ilustra en la **figura 3.9**.
- Un conjunto especializado de instrucciones permite acceder a información almacenada en la **memoria**, realizando a continuación sobre ellas **operaciones aritméticas o lógicas**. Los resultados intermedios pueden ser devueltos nuevamente a la memoria, o almacenados en **registros temporales**, para ser utilizados posteriormente. La **figura 3.10** muestra este tipo de instrucción.

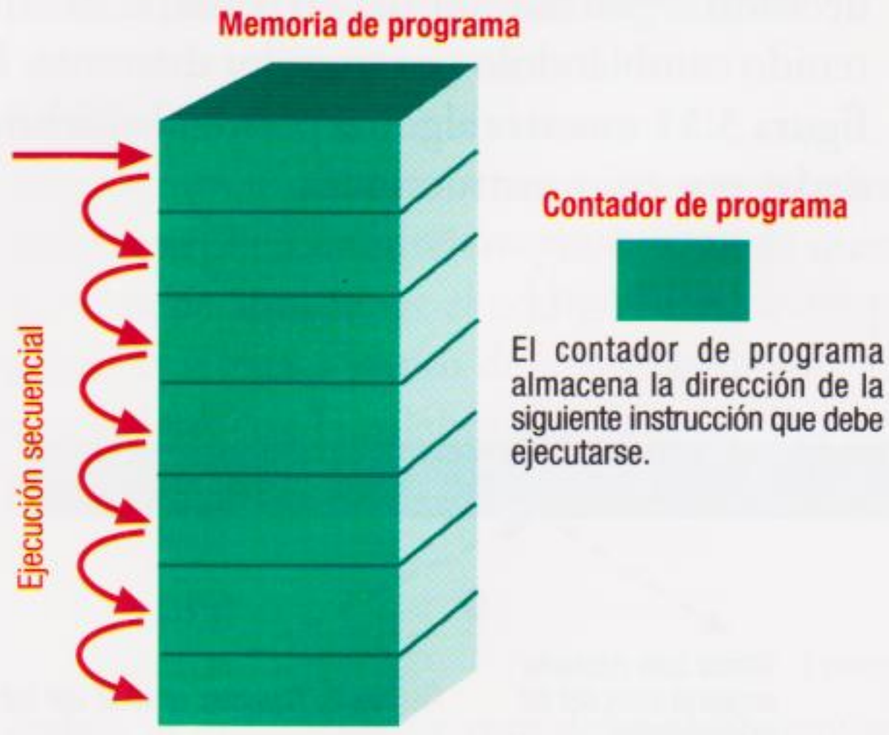


Figura 3.8 Flujo probable en la ejecución de un programa

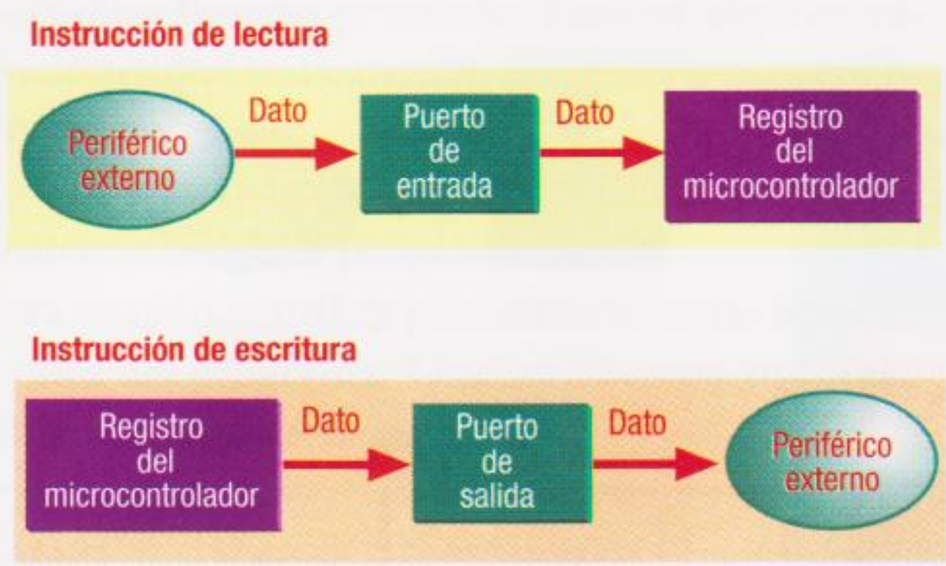


Figura 3.9 *Instrucciones de entrada/salida.*

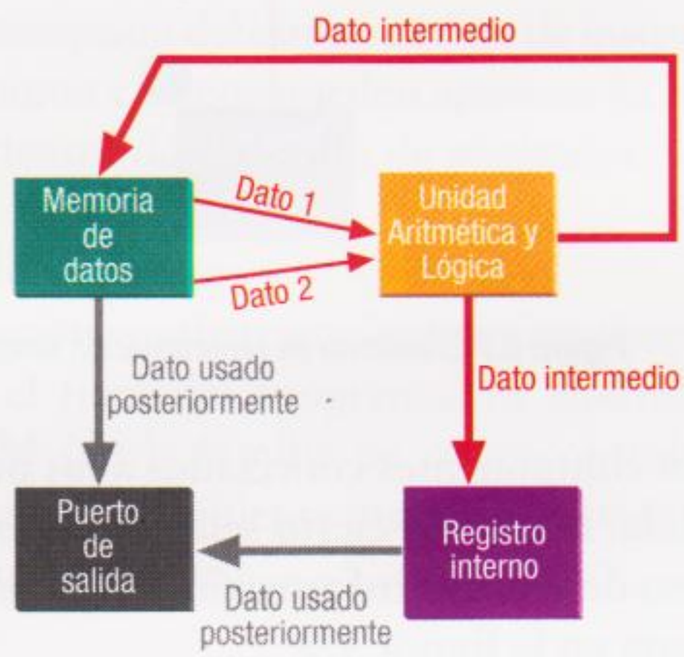


Figura 3.10 *Instrucciones de transferencia a memoria y cálculos aritméticos y lógicos.*

- En ocasiones es conveniente manipular directamente los *bits* individuales de los datos almacenados en la memoria o disponibles en un registro interno. El microcontrolador dispone de instrucciones que aislan cada *bit* de manera individual. De este modo se puede tomar una decisión según el valor del *bit* o alterar su contenido cambiándolo por un valor diferente. La **figura 3.11** muestra algunas posibilidades brindadas por estas instrucciones.

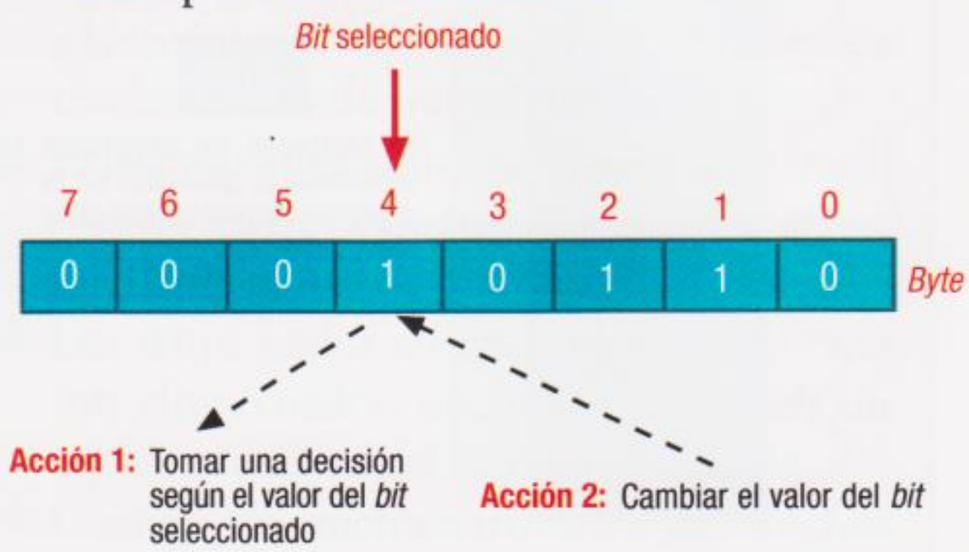


Figura 3.11 *Instrucciones de manipulación de bits*

- Los programas no siempre se ejecutan de manera lineal; algunas veces es necesario alterar el contenido del contador de programa, lo que deriva en una ejecución no secuencial de las instrucciones. Esta característica confiere una mayor flexibilidad a los programas. Las instrucciones de salto se dividen en dos categorías. La primera de ellas corresponde a las **instrucciones de salto condicional**, en las cuales el salto se produce si, y sólo si se cumple una determinada condición.

La segunda categoría corresponde a las **instrucciones de salto incondicional**. Como su nombre lo indica, no se requiere que se cumpla ninguna condición previa para generar el salto. La **figura 3.12** muestra los dos tipos de salto indicados.

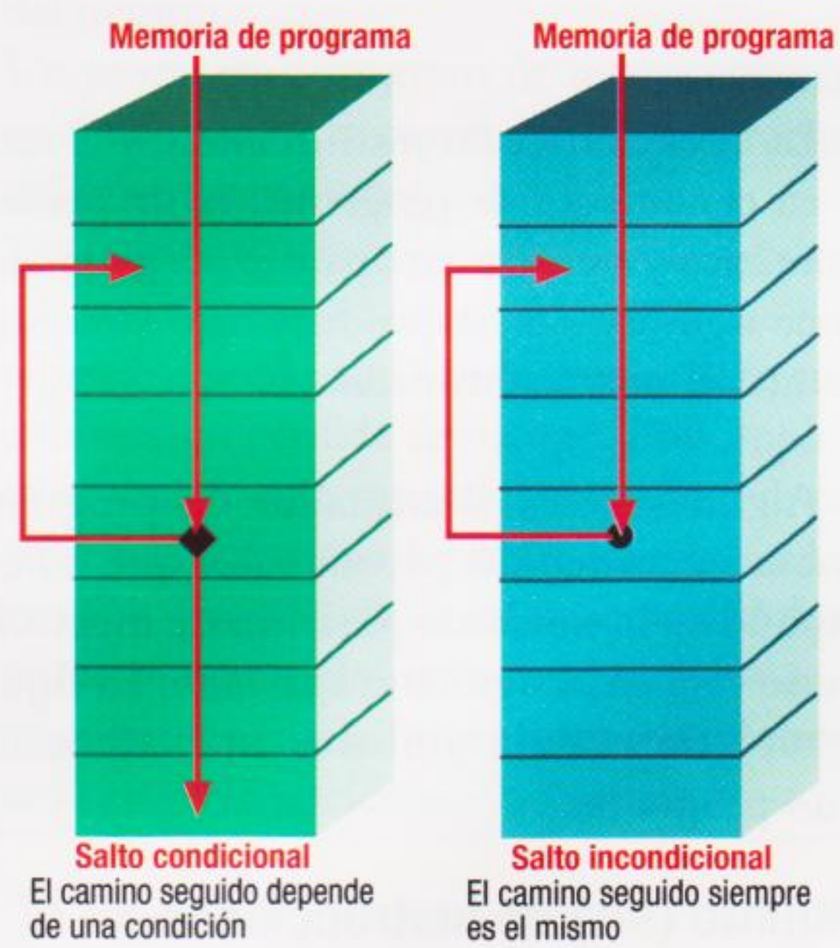


Figura 3.12 *Salto condicional e incondicional*

- En ocasiones, algunos procesos del microcontrolador no se pueden encasillar en ninguno de los grupos mencionados. Las instrucciones implicadas en la ejecución de estos procesos se conocen con el nombre de **instrucciones de control**, y su presencia ayuda a mejorar de manera significativa el funcionamiento de los programas. A título de ejemplo, podemos citar las instrucciones que habilitan (o inhiben) las interrupciones externas. En su momento se mostrará un panorama completo de ellas.

Instrucciones de desplazamiento de *bits*

En ocasiones es conveniente desplazar los *bits* de un registro (o posición de memoria) para ejecutar una operación matemática sencilla, o simplemente para crear un efecto que pueda reflejarse en el medio de salida, como es el caso de un *display* o despliegue luminoso o un conjunto de diodos LED.

Para iniciar el tema, vamos a plantear el siguiente problema: se debe multiplicar el contenido del registro W con el literal 2, reemplazando el contenido previo del registro con el valor obtenido en la multiplicación.

W * 2 ——> W

En el esquema mostrado en la **figura 3.13**, el registro W contiene el valor 24. La respuesta (48) reemplaza el contenido previo del registro W.

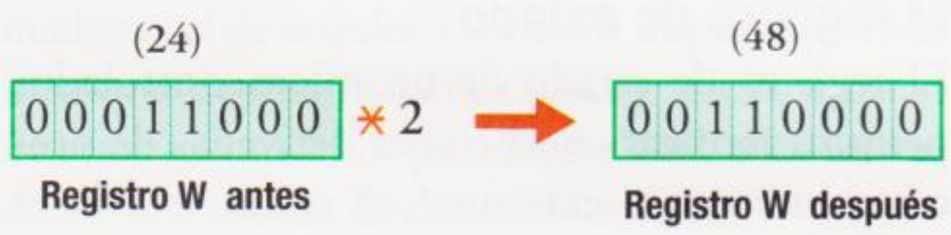


Figura 3.13 *El contenido del registro W multiplicado por 2*

Si compara el contenido del registro W antes y después de la multiplicación, verá que la diferencia radica en la posición que ocupa cada *bit*: es suficiente desplazar el contenido original del registro W una posición a la izquierda, para obtener el resultado de multiplicar el número por 2 (**figura 3.14**).

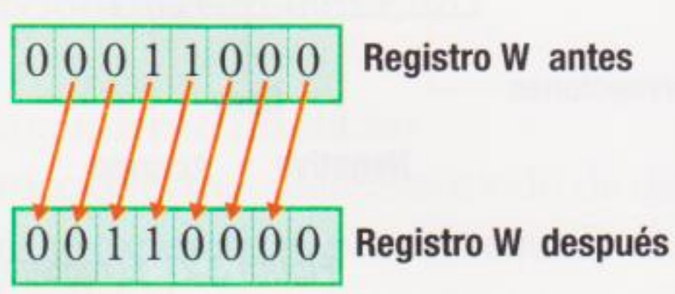


Figura 3.14 *Desplazamiento a la izquierda*

> Desplazar el contenido de un registro un *bit* a la izquierda, es equivalente a multiplicar su contenido por 2.

En cuanto a la operación de división, el proceso es a la inversa: basta con desplazar el contenido del registro un *bit* a la derecha, para obtener como resultado la división del número entre dos. Observe el proceso en la **figura 3.15**.

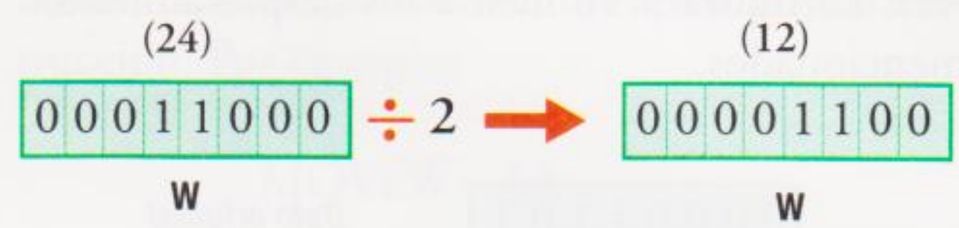


Figura 3.15 *Desplazamiento a la derecha*

¿Qué ocurre con el *bit* que sale del registro, tanto a izquierda como a derecha, después del desplazamiento? Este se debe almacenar en un sitio preparado especialmente para ello.

El microcontrolador posee un registro especial, denominado **registro de estado**, para el cual el valor de los *bits* cambia según el resultado de las operaciones ejecutadas. La **figura 3.16** muestra el uso del registro de estado para almacenar el *bit* excedente que resulta de un desplazamiento a la derecha.

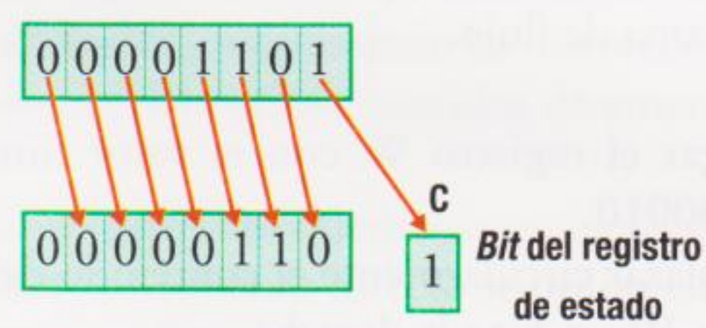


Figura 3.16 *El registro de estado del microcontrolador*

Ahora bien, ¿cuál es el valor del *bit* que ingresa por el extremo más a la izquierda del registro en un desplazamiento hacia la derecha? Usualmente, es el valor cero. Esto se aplica también al desplazamiento en sentido contrario, en cuyo caso el *bit* que ingresa por el extremo más a la derecha es el dígito cero. La **figura 3.17** ilustra este proceso.

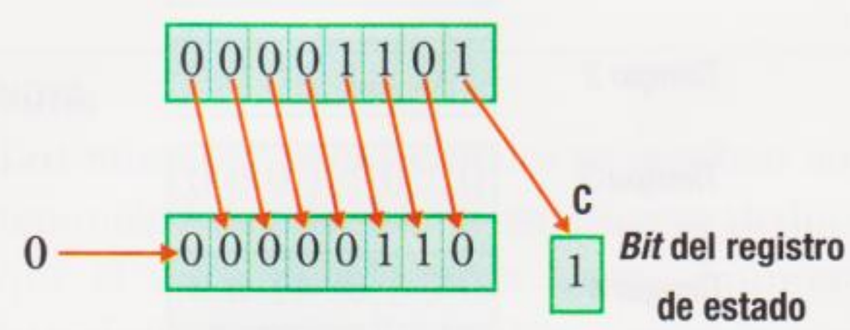


Figura 3.17 *Bits que ingresan al registro*

El desplazamiento de *bits* permite llevar a cabo tareas diferentes a las que se derivan del cálculo numérico: es posible desplazar circularmente el contenido de un registro, de modo que el *bit* más a la derecha se inserta como primer *bit* más a la izquierda (o en sentido inverso, según se desee). La **figura 3.18** ilustra los desplazamientos mencionados.

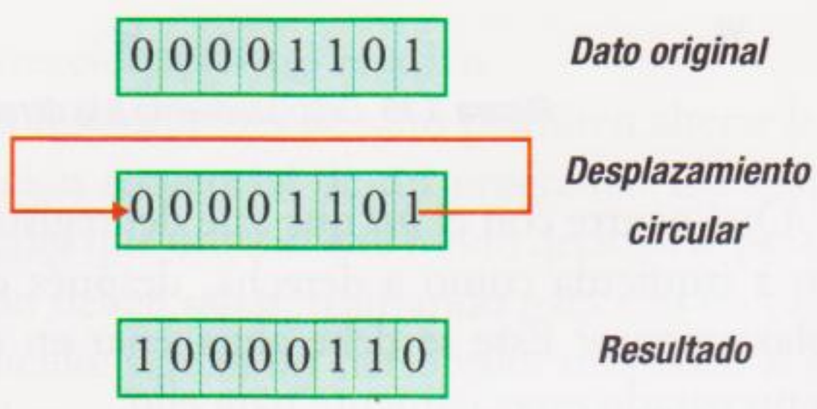


Figura 3.18 *Desplazamiento circular*

Aplicación del desplazamiento circular

Estudie el siguiente programa: en el encontrará la solución a un problema de animación visual sobre un dispositivo óptico. La **figura 3.19** muestra el resultado de este proceso y la **figura 3.20** su diagrama de flujo.

1. Cargar el registro W con el valor binario 11000010.
2. Desplazar circularmente el contenido del registro W un *bit* a la derecha.
3. Enviar al puerto de salida el dato almacenado en el registro W.
4. Hacer una pausa.
5. Regresar al paso 2, con lo que el proceso se repite indefinidamente.

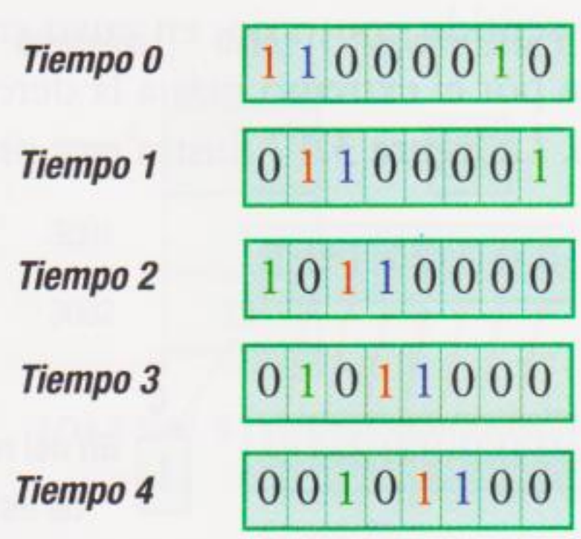


Figura 3.19 *Aplicación del desplazamiento circular*

Figura 3.20 *Diagrama de flujo*

El registro de estado

El registro de estado de un microcontrolador permite verificar condiciones derivadas de procesos internos o externos, así como establecer condiciones especiales de control. Cada *bit* en el registro de estado tiene un significado particular el cual puede variar según la marca y familia del microcontrolador. A continuación vamos a describir los *bits* más importantes, dejando la explicación de los restantes para cuando corresponda según las aplicaciones desarrolladas. En la **figura 3.21** se muestra una parte de la estructura típica de un registro de estado.

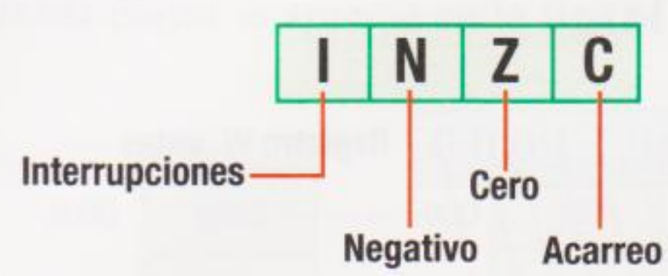


Figura 3.21 *El registro de estado*

A continuación describimos el significado de los *bits* de control.

C - **Control de acarreo.** Toma el valor 1 cuando se produce un *bit* de acarreo (*Carry*) en una suma de números binarios.

Z - **Control de cero.** Toma el valor 1 cuando el resultado de una operación es cero.

N - **Control de número negativo.** Toma el valor 1 cuando el resultado de una operación es negativo.

I - **Activación global de interrupciones.** Al establecer su valor a 1 se activa la detección de interrupciones: el microcontrolador puede ser interrumpido en su funcionamiento por elementos externos.

El microcontrolador dispone de instrucciones que permiten tomar decisiones de acuerdo con el valor presente en los *bits* de estado. Estas instrucciones son útiles para crear ciclos o bifurcar a otras posiciones de memoria.

Modos de direccionamiento

Uno de los problemas al que se enfrenta un programador es el de acceder a los datos disponibles en la memoria de una manera rápida y eficaz. Existen diversos mecanismos para lograr esto, a los que se denomina **modos de direccionamiento.**

Los modos de direccionamiento más conocidos son los siguientes:

- Direccionamiento implícito
- Direccionamiento inmediato
- Direccionamiento directo
- Direccionamiento relativo
- Direccionamiento indexado
- Direccionamiento extendido
- Direccionamiento indirecto

Direccionamiento implícito

Las instrucciones que usan este modo de direccionamiento se caracterizan por no requerir el uso de operandos, debido a que no necesitan acceder a la memoria de datos. Por ejemplo, la instrucción

```
NOP
```

pertenece a esta categoría. La instrucción NOP no produce ningún efecto visible en la ejecución del programa, a pesar de lo cual es útil porque consume cierta cantidad de ciclos de máquina y por ello permite la creación de retardos de tiempo.

Direccionamiento inmediato

Se presenta cuando el dato no proviene de la memoria, sino que está incluido en la misma instrucción. Por ejemplo:

```
MOVLW   5A
```

Esta instrucción almacena el literal 5A en el registro W. Se puede apreciar que el dato no procede del exterior ni tampoco de la memoria de datos. La **figura 3.22** muestra el efecto del direccionamiento inmediato.

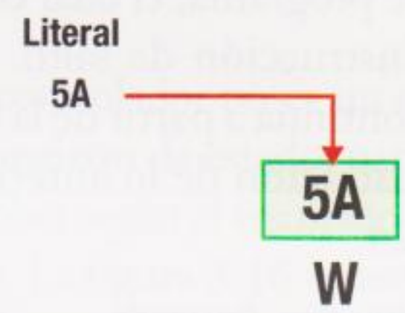


Figura 3.22 *Direccionamiento inmediato*

El direccionamiento inmediato se utiliza cuando los datos que toman las instrucciones son literales (constantes) conocidos de antemano.

Direccionamiento directo

El direccionamiento directo se utiliza cuando el dato se transfiere hacia, o desde, una posición de memoria particular. Por ejemplo:

```
MOVWF   0x0C
```

Esta instrucción mueve el contenido del registro W hacia la posición de memoria 0x0C. El contenido de la posición 0x0C desaparece y en su lugar se coloca el valor del registro W. La **figura 3.23** ilustra el direccionamiento directo.

NOTA:
Los números hexadecimales se escriben anteponiéndoles 0x. De lo anterior se deduce que el valor 0x0C puede leerse: "número hexadecimal 0C". Este valor corresponde al número decimal 12.

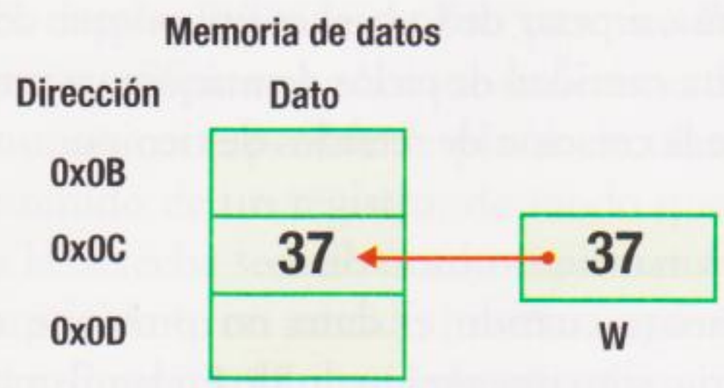


Figura 3.23 Direccionamiento directo

Direccionamiento relativo

Las instrucciones de salto permiten alterar la ejecución secuencial de un programa. Las instrucciones que utilizan este modo deciden a que posición deben saltar, realizando para ello un cálculo sencillo: suman un cierto valor numérico al valor del contador de programa, el cual contiene la dirección de la instrucción de salto. La ejecución del programa continúa a partir de la dirección calculada. Una ilustración de lo anterior se observa en la **figura 3.24**.

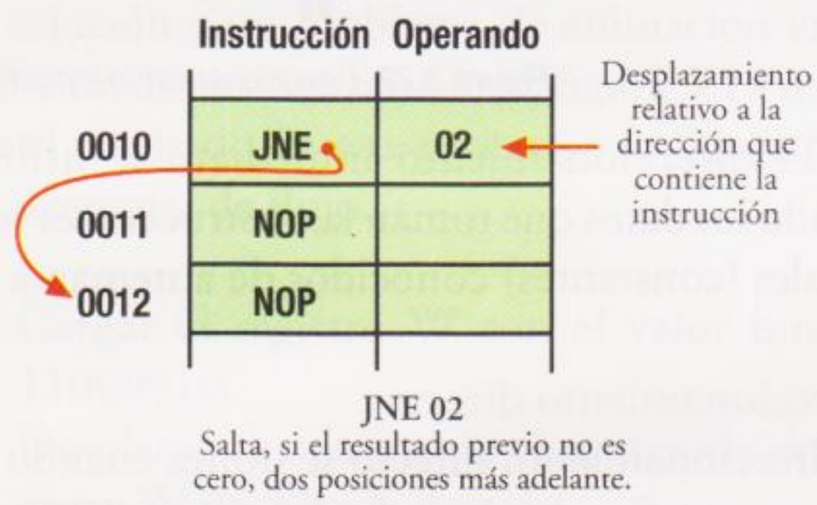


Figura 3.24 Direccionamiento relativo

Direccionamiento indexado

La dirección de destino se calcula utilizando como base un registro especial. En la **figura 3.25** se muestra el caso en consideración.

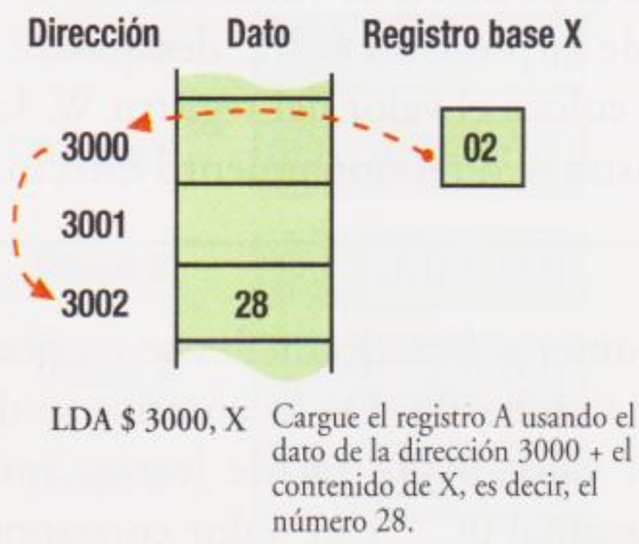


Figura 3.25 Direccionamiento indexado

Direccionamiento extendido

El direccionamiento extendido permite acceder a todo el espacio de memoria del microcontrolador. La dirección se almacena en dos o tres *bytes* del operando, de modo que sin importar el tamaño de la dirección destino se puede acceder directamente a ella.

La **figura 3.26** ilustra el modo de direccionamiento extendido.

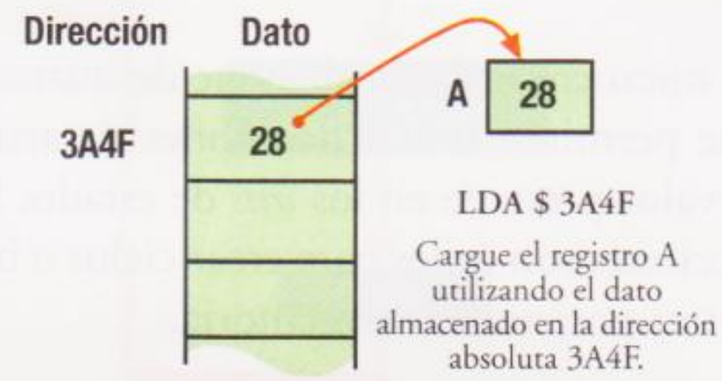


Figura 3.26 Direccionamiento extendido

Direccionamiento indirecto

En el modo de direccionamiento indirecto la dirección de una posición de memoria se calcula mediante una doble referencia.

En primer lugar se obtiene el dato contenido en una cierta posición de memoria, y con base en este dato, se obtiene la dirección efectiva de la posición de memoria deseada.

La dirección efectiva es la dirección deseada. En el modo de direccionamiento indirecto esta se calcula como se aprecia en la **figura 3.27**.

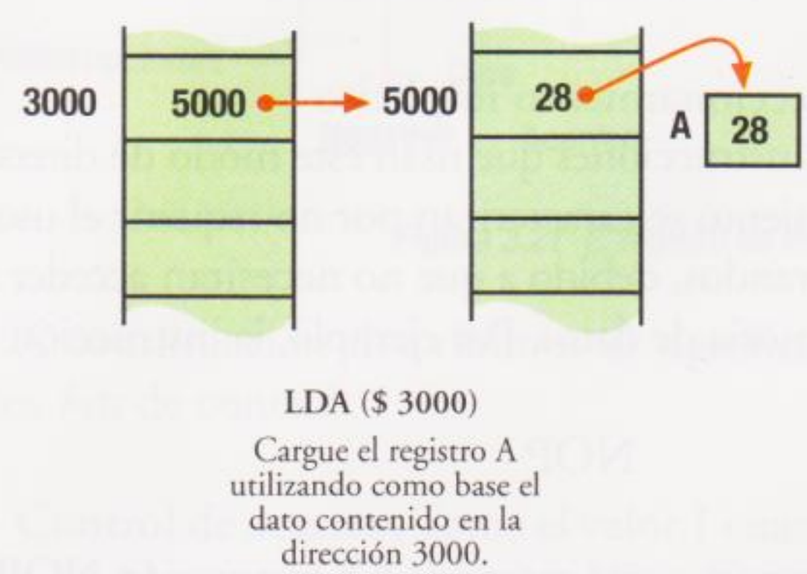
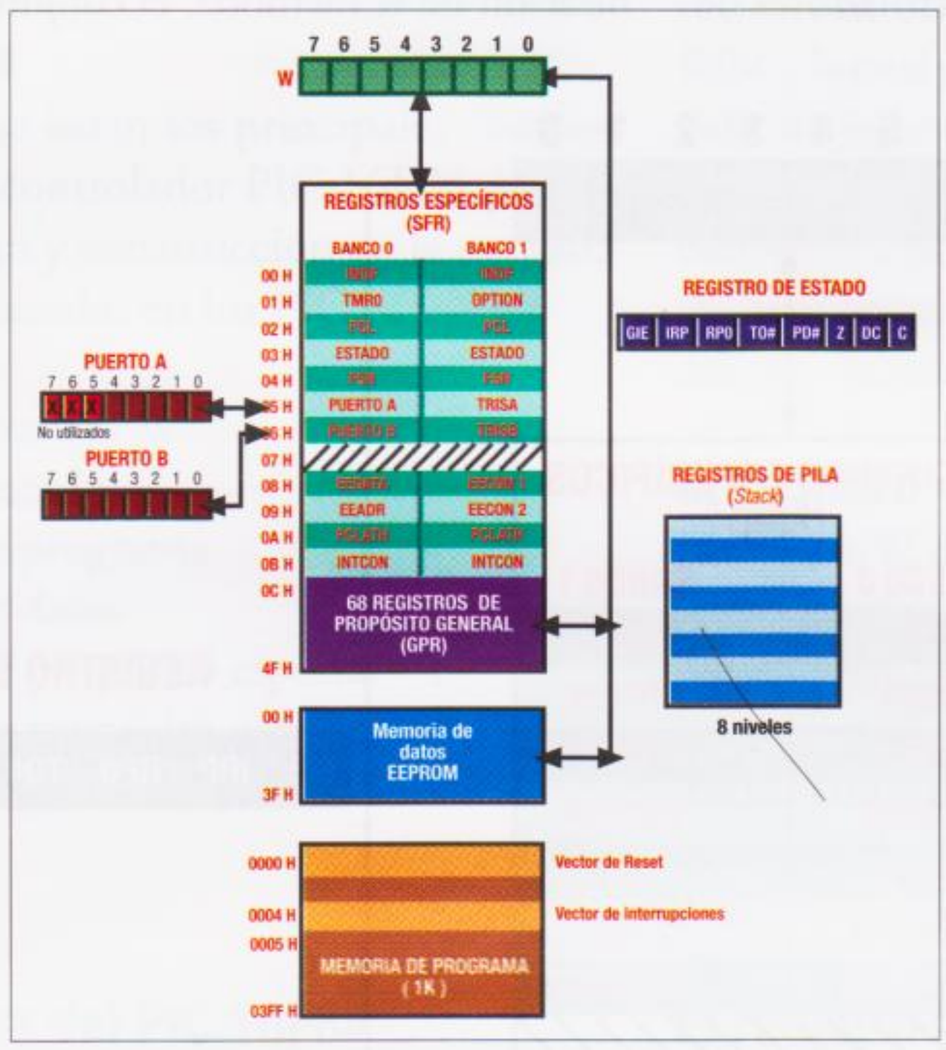


Figura 3.27 Direccionamiento indirecto

LECCIÓN 4



MODELO DE PROGRAMACIÓN PARA LOS MICROCONTROLADORES PIC

Antes de elaborar cualquier programa, se debe conocer el modelo de programación o estructura lógica de los principales componentes de un microcontrolador. En esta lección veremos el del PIC 16F84, el cual puede ser aplicado, con algunas pequeñas variaciones propias de cada familia, a los otros modelos fabricados por Microchip.

Vamos a iniciar este tema estudiando las características lógicas del PIC 16F84 de Microchip, un microcontrolador versátil y de arquitectura simple. Estos conceptos se pueden aplicar en su gran mayoría a los demás microcontroladores PIC. Más adelante, cuando veamos la programación de otros microcontroladores de diferentes modelos, explicaremos los detalles específicos de cada uno.

Esta lección presenta el estudio sistemático de los elementos que todo programador debe considerar a la hora de elegir un microcontrolador: la organización de la memoria, el conjunto de instruccio-

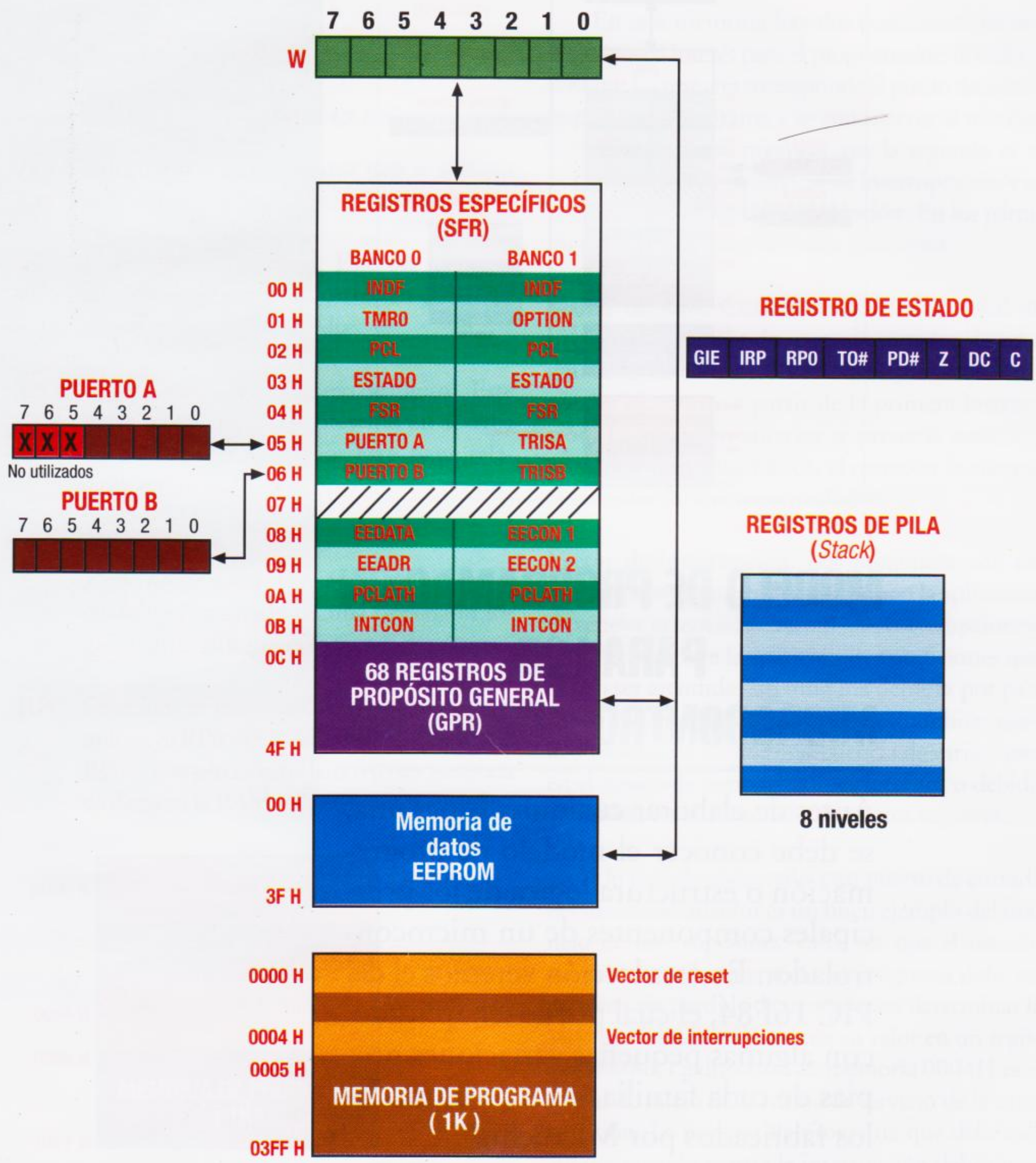


Figura 4.1 *La arquitectura del PIC 16F84 desde el punto de vista del programador*

nes disponible, los modos de direccionamiento y las particularidades generales del sistema elegido. Sin embargo, estudiaremos estos elementos desde la óptica del programador, a quien le importa en primera instancia, sólo la naturaleza lógica de los componentes y no su comportamiento eléctrico.

Los elementos

A continuación se listan los principales componentes del microcontrolador PIC 16F84, implicados en el diseño y construcción de la mayoría de aplicaciones basadas en los PIC.

- El registro de trabajo W
- El registro de estado
- La memoria de programa
- La memoria de datos
- Los registros de propósito especial
- Los registros de propósito general
- Los registros de pila (*stack*)
- El puerto A
- El puerto B

La arquitectura del PIC 16F84 (La visión del programador)

Un programador concibe la arquitectura del PIC 16F84 como un conjunto de recursos disponibles, al que puede acceder mediante el uso sistemático de un conjunto de instrucciones. El orden con el que escribe las instrucciones influye en el funcionamiento del sistema; es por este motivo que el programador debe conocer con todo detalle el comportamiento lógico de los diferentes elementos que conforman la arquitectura del microcontrolador. La **figura 4.1** muestra la apariencia que presenta el PIC 16F84 a un programador. En los siguientes apartados veremos cada elemento en detalle.

Los microcontroladores de la familia Microchip utilizan la arquitectura Harvard, la que se caracteriza por independizar el bus de datos del bus de memoria. Esto permite adaptar el tamaño de los buses de acuerdo con los requerimientos del sistema, además de aumentar sensiblemente la velocidad de trabajo. Adicionalmente, los microcontroladores PIC se basan en la tecnología RISC, la cual se caracteriza por un conjunto reducido de instrucciones que se ejecuta en un sólo ciclo de máquina. Las tecnologías mencionadas confieren a este modelo de microcontrolador una enorme potencia en comparación con sus competidores, en cuanto a la relación costo/beneficio.

Figura 4.2 *El registro de trabajo W*

El registro de trabajo W

El registro W, de 8 *bits*, sirve para almacenar un dato, usualmente de forma temporal, cuyo valor será utilizado posteriormente en un cálculo matemático o en la transferencia entre registros y memoria (**figura 4.2**). Toda la información debe pasar por este registro, razón por la cual es de suma importancia para la mayoría de instrucciones del microcontrolador.

El registro de estado (*status register*)

La mayoría de los procesos ejecutados por el microcontrolador generan efectos que pueden repercutir sobre el desenvolvimiento del programa. Como ejemplo de lo anterior, considere la instrucción que desplaza a la derecha el contenido de una posición de memoria. Esta instrucción puede dar como resultado, en algún instante del tiempo, un valor igual a cero. En ocasiones, esta situación puede ser de interés para el programador.

El *bit* Z (cero) del registro de estado toma el valor 1 si el contenido de la última operación efectuada es cero. En este caso, el uso de una instrucción apropiada permite al programa verificar el valor del *bit* y bifurcar a una posición de memoria no secuencial, continuando desde allí la ejecución del programa.

La **figura 4.3** muestra la estructura del registro de estado. Los campos en color gris representan estados cuyo uso se relaciona con problemas de mayor complejidad que no aplicaremos por ahora.

Figura 4.3 *El registro de estado*

Los *bits* del registro de estado que más se utilizan son los siguientes:

C *Carry* (*bit* de acarreo). Este *bit* se activa cuando se presenta un acarreo en el *bit* más significativo del resultado.

DC *Digit Carry* (*bit* de acarreo de dígito). Este *bit* se activa cuando se presenta un acarreo en el *bit* 3 del resultado. Esto tiene relación, sobre todo, con operaciones en formato BCD.

Z *Zero* (*bit* de cero). Este *bit* toma el valor 1 cuando el resultado de una operación lógica o aritmética es cero.

RP0 Selecciona el banco de memoria que se va a utilizar. Si RP0 = 0, se selecciona el banco 1; si RP0 = 1, se selecciona el banco 0 (ver memoria de datos en la RAM un poco más adelante).

Figura 4.4 *La memoria de programa*

La memoria de programa

En el PIC 16F84 los programas de usuario se almacenan en una zona de memoria con un tamaño de 1K. Esto es suficiente para albergar la solución de buena parte de los problemas de control. La **figura 4.4** muestra un esquema básico de la memoria de programa.

En esta memoria hay dos posiciones que tienen especial interés para el programador: 0000H y 0004H. La primera corresponde al punto de arranque de todo programa y se conoce con el nombre de vector de *reset*, mientras que la segunda es el inicio de la rutina de servicio de interrupciones y se la denomina **vector de interrupción**. En los párrafos siguientes se amplían estos conceptos.

Vector de *reset*. Cuando se aplica una señal de nivel bajo a la línea de *reset*, el contador de programa toma el valor 0000H y el programa empieza a ejecutarse a partir de la primera instrucción. La situación anterior se presenta también, pero de forma automática, al conectar la energía al circuito del microcontrolador.

Vector de interrupción. Los programas con un cierto grado de complejidad admiten la aplicación de señales externas, denominadas **interrupciones**, que alertan sobre la aparición de condiciones que deben ser atendidas sin ninguna demora por parte del sistema. Las interrupciones también pueden producirse como consecuencia de variaciones en el estado de temporizadores internos o debido al cambio en el contenido de ciertos registros.

Un teclado conectado a un puerto de entrada del microcontrolador es un buen ejemplo del manejo de interrupciones: cada vez que el usuario del sistema pulsa una tecla, el programa debe reaccionar sin pérdida de tiempo para determinar la tecla presionada y almacenar su valor en un registro interno. La dirección de memoria 0004H es el punto de inicio de la rutina de servicio de la interrupción. La porción de programa que debe ejecutarse cuando ocurra la interrupción debe guardarse a partir de la dirección 0004H.

De lo expuesto en los párrafos anteriores tenemos dos situaciones posibles:

- El programa no contiene rutina de manejo de interrupciones, por lo que se puede utilizar toda la memoria de manera lineal para almacenar el programa de usuario, **figura 4.5a**. Esto en el caso del 16F84. En otros modelos este manejo puede ser diferente.

- El programa incluye una rutina de servicio de interrupciones, lo que obliga a la inclusión de un salto al comienzo del programa (dirección 00H) para omitir el código de programa que atiende la rutina de servicio, **figura 4.5b**.

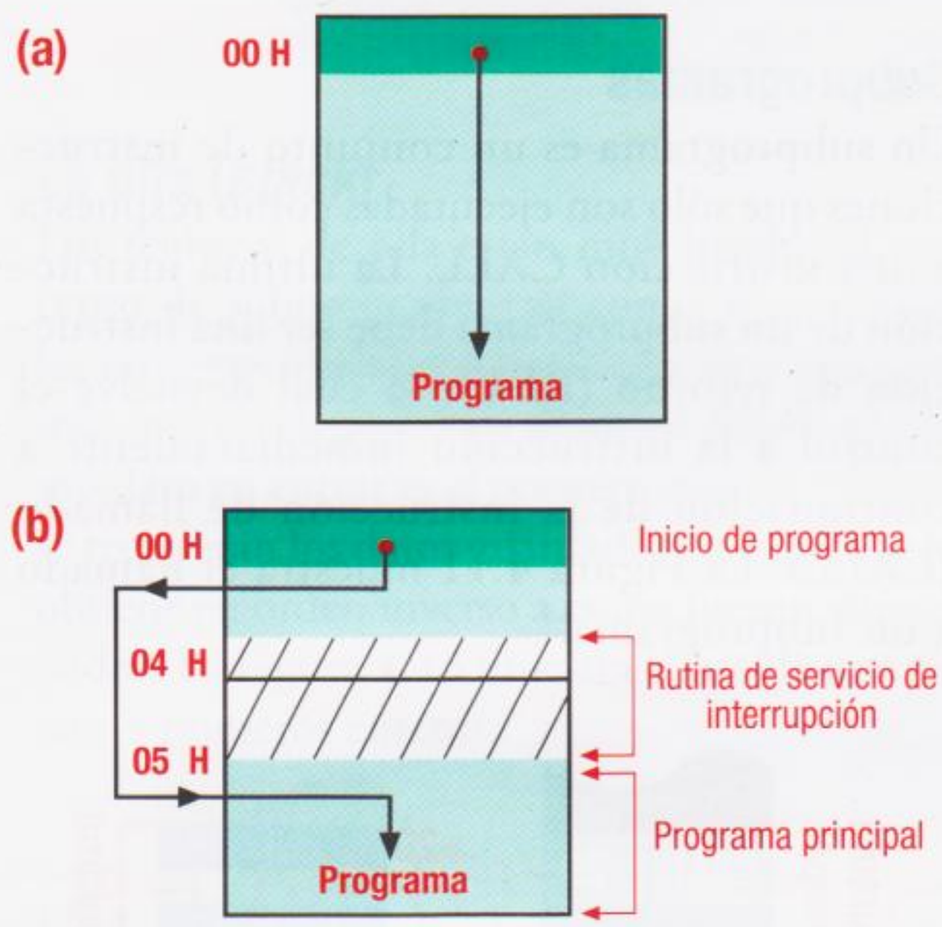


Figura 4.5 *La rutina de servicio de interrupciones*

La memoria de datos en RAM

La memoria de datos en RAM del PIC 16F84 está compuesta por 64 registros almacenados en posiciones de memoria consecutivas. Los primeros 12 registros, comprendidos entre 00H y 0BH, cumplen un propósito específico en el contexto de funcionamiento del microcontrolador. Los últimos 48 registros, en el rango comprendido entre 0CH y 2FH, se usan para almacenar información temporal. La **figura 4.6** muestra un esquema simplificado de la memoria de datos.

Figura 4.6 *Esquema simplificado de la memoria de datos*

Los registros de propósito especial

El PIC 16F84 se caracteriza por disponer de dos bancos de memoria de datos: el banco cero y el banco uno. Lo anterior implica que cada registro está disponible tanto en el banco cero como en el banco uno. La **figura 4.7** muestra los doce primeros registros (00H-0BH) y su significado dependiendo del banco de memoria utilizado.

	BANCO 0	BANCO 1	
00 H	INDF	INDF	Direccionamiento indirecto
01 H	TMR0	OPTION	Temporizador/contador
02 H	PCL	PCL	Parte baja del contador de programa
03 H	ESTADO	ESTADO	Registro de estado
04 H	FSR	FSR	Selector de registros
05 H	PUERTO A	TRISA	Puerto A
06 H	PUERTO B	TRISB	Puerto B
07 H			
08 H	EEDATA	EECON 1	Acceso a la memoria EEPROM
09 H	EEADR	EECON 2	
0A H	PCLATH	PCLATH	Parte alta del contador de programa
0B H	INTCON	INTCON	Control de interrupciones

Figura 4.7 *Los registros de propósito especial*

La forma de uso de los registros de propósito especial se verá en detalle cuando estudiemos el conjunto (*set*) de instrucciones del microcontrolador y la programación de los *bits* del registro de estado.

Los registros de propósito general

Los 36 registros de propósito general (00H-2FH) están **mapeados** sobre el banco de memoria cero; esto significa que sólo uno de los registros, el correspondiente al banco cero, estará disponible para el programador. La **figura 4.8** muestra la disposición de los registros de propósito general.

Figura 4.8 *Los registros de propósito general*

Los registros de pila (*stack*)

Es bastante común dividir un programa en pequeñas porciones de código, cada una de las cuales cumple un propósito especial. El programa principal, es decir, aquel que será ejecutado cuando el microcontrolador reciba una señal de *reset* (o cuando sea energizado) efectuará llamadas a estos subprogramas en diferentes puntos, de acuerdo con las necesidades establecidas.

Un subprograma debe terminar con una instrucción de retorno. Esta instrucción deposita en el contador de programa la dirección de memoria en donde debe reanudarse la ejecución del programa. El microcontrolador utiliza para conseguir este propósito un conjunto de registros internos denominados registros de pila (*stack*). El conjunto de registros de pila está conformado por ocho registros (**figura 4.9**).

Figura 4.9 *Los registros de pila (stack)*

En cada nueva llamada a un subprograma se almacena la dirección de retorno en el siguiente registro de la pila, lo que permite regresar al sitio correcto cada vez que finalice cada subprograma. Debe hacerse notar que el programador no puede acceder directamente al contenido de ninguno de los registros de la pila.

La memoria EEPROM de datos

El PIC 16F84 dispone de una memoria EEPROM que consta de 64 *bytes*, comprendidos entre 00H y 3FH (**figura 4.10**). Para acceder a estos registros hay que utilizar los registros de control EEDATA, EEADR, EECON1 y EECON2. En capítulos posteriores veremos la forma de aplicar la programación de estos registros.

Figura 4.10 *La memoria EEPROM de datos*

Subprogramas

Un subprograma es un conjunto de instrucciones que sólo son ejecutadas como respuesta a una instrucción CALL. La última instrucción de un subprograma debe ser una instrucción de retorno (RET), la cual devuelve el control a la instrucción inmediatamente a continuación de la instrucción de llamado (CALL). La **Figura 4.11** muestra el llamado a un subprograma.

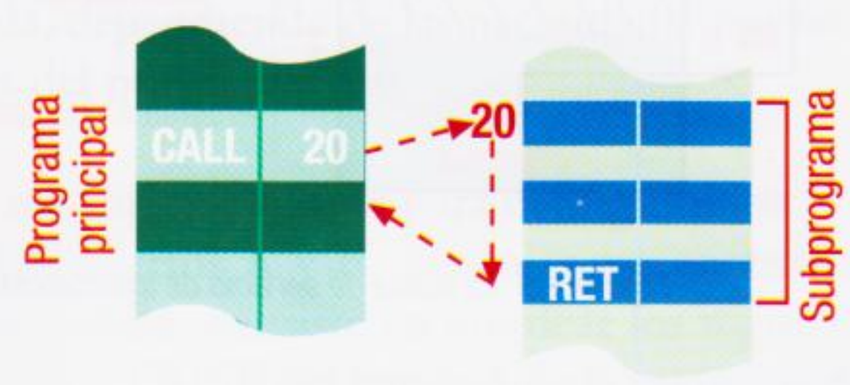


Figura 4.11 *Llamado a un subprograma*

¿Por qué son útiles los subprogramas?

Imagine que elige un microcontrolador que no posea la instrucción para multiplicar dos números enteros. Un método para resolver este problema consiste en idear un conjunto de instrucciones que realicen la multiplicación a partir del contenido de dos registros preestablecidos, dejando la respuesta en un tercer registro.

El uso más inteligente de las instrucciones que multiplican dos números enteros consiste, no en agregarlas cada vez que se deban multiplicar dos números, sino en escribirlas aparte (fuera del programa principal) y llamarlas cada vez que se las requiera. Con esta idea en mente observe la **Figura 4.12** y siga la dirección de las flechas.

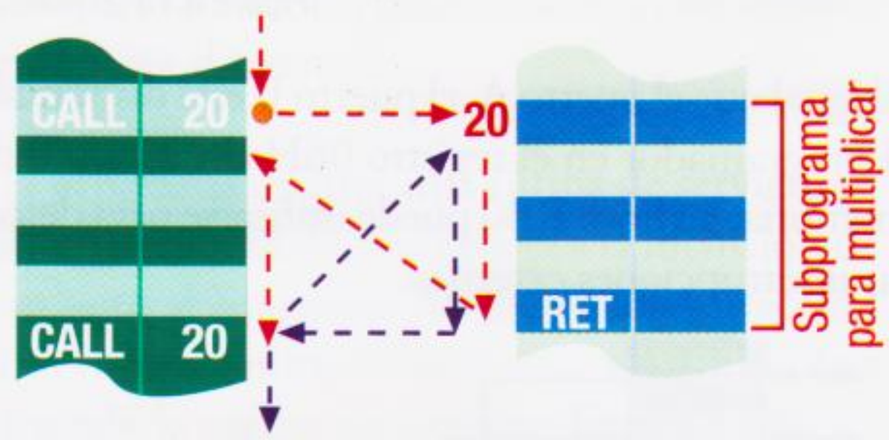


Figura 4.12 *Un subprograma para multiplicar dos números enteros*

La pila (*stack*)

Los registros de pila están muy ligados al concepto de subprograma (así como al manejo de las interrupciones). En principio, una pila no es más que un depósito de datos en donde el último dato en entrar es el primero en salir. Cuando se recuperan los datos guardados en la pila, se los obtiene en orden inverso a como fueron almacenados. La **Figura 4.13** muestra el uso de una pila con 3 números enteros.

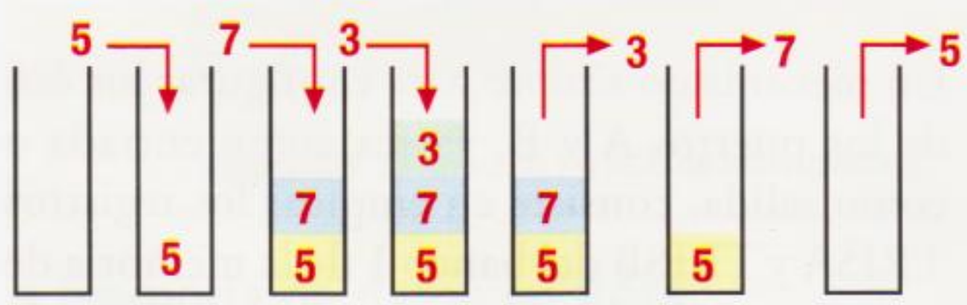


Figura 4.13 *Funcionamiento de una pila*

En el siguiente apartado veremos la relación que existe entre las pilas y los subprogramas.

Funcionamiento de los registros de pila del microcontrolador

Para entender de manera cabal el funcionamiento de los registros de pila, considere un ejemplo simple de llamado a subprograma. El programa mostrado en la **Figura 4.14** hace un llamado al subprograma rotulado con la etiqueta **SubA**. Antes de que el contador de programa modifique su contenido por el de la instrucción CALL, es necesario preservar la dirección de retorno. Esto permitirá que, una vez finalizado el subprograma, el microcontrolador continúe la ejecución a partir de la siguiente instrucción en donde se produjo el llamado.

Figura 4.14 *Llamado a un subprograma*

La instrucción CALL guarda en el fondo de la pila la dirección de retorno, la cual corresponde a la dirección de la siguiente instrucción.

En la **Figura 4.15**, a grandes rasgos, se observa la estructura del subprograma **SubA**. Este subprograma contiene un llamado a un segundo subprograma, **SubB**, lo que genera un dato adicional en la cima de la pila, antes de proceder a modificar el contenido del contador de programa.

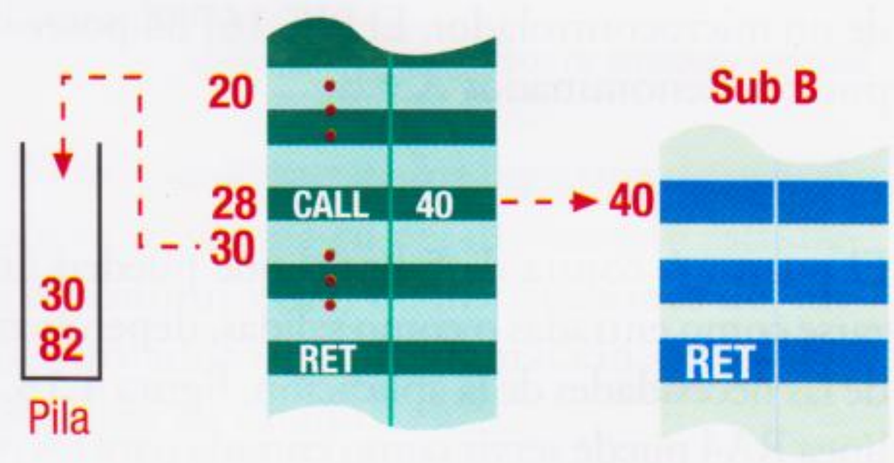


Figura 4.15 *Llamando a un subprograma desde otro subprograma.*

Cuando se ejecuta la instrucción RET del segundo subprograma (**SubB**), la unidad de control del microcontrolador extrae el dato almacenado en el tope de la pila (el cual corresponde a la dirección de retorno) y lo deposita en el contador de programa. El programa continúa ejecu-

tandose después de la instrucción CALL del primer subprograma, **figura 4.16**.

Figura 4.16 *Retorno del segundo subprograma*

La ejecución de la instrucción RET del primer subprograma causa un retorno al programa principal. Observe el contenido de la pila, **Figura 4.17**.

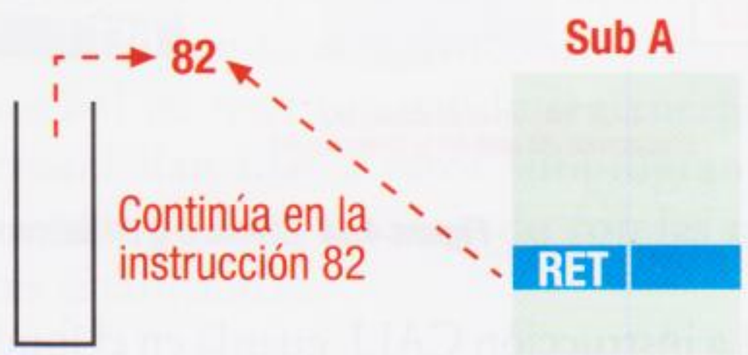


Figura 4.17 *Retorno del primer subprograma*

Los procesos relacionados con el manejo de la pila son transparentes al programador, es decir, ocurren de manera automática en el interior del microcontrolador.

Puertos

Recordemos que un puerto es un elemento a través del cual se introduce o se extrae información de un microcontrolador. El PIC 16F84 posee dos puertos denominados A y B.

El puerto A

El puerto A consta de 5 líneas que pueden utilizarse como entradas o como salidas, dependiendo de las necesidades de la aplicación, **figura 4.18**. La línea RA4 puede servir como entrada para los pulsos de reloj aplicados al temporizador interno TMR0. El programador puede acceder al contenido del puerto leyendo (o escribiendo) el contenido del registro 05H del banco 0 de memoria.

El puerto B

El puerto B está conformado por 8 líneas que pueden configurarse como entrada o como salida según los requerimientos de la aplicación, **figura 4.19**.

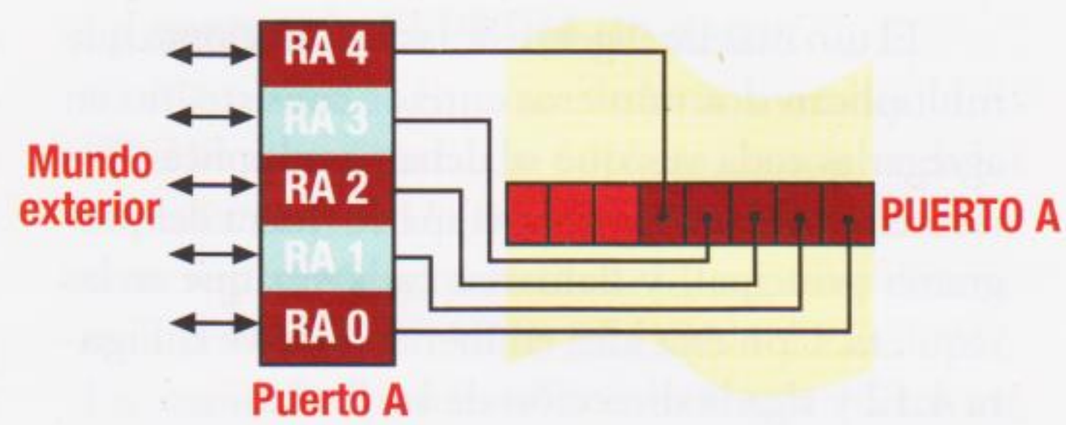


Figura 4.18 *El puerto A*

Al igual que el puerto A, el puerto B está disponible al programador en el registro 06H del banco 0 de memoria. La línea RB0 puede utilizarse para detectar interrupciones externas.

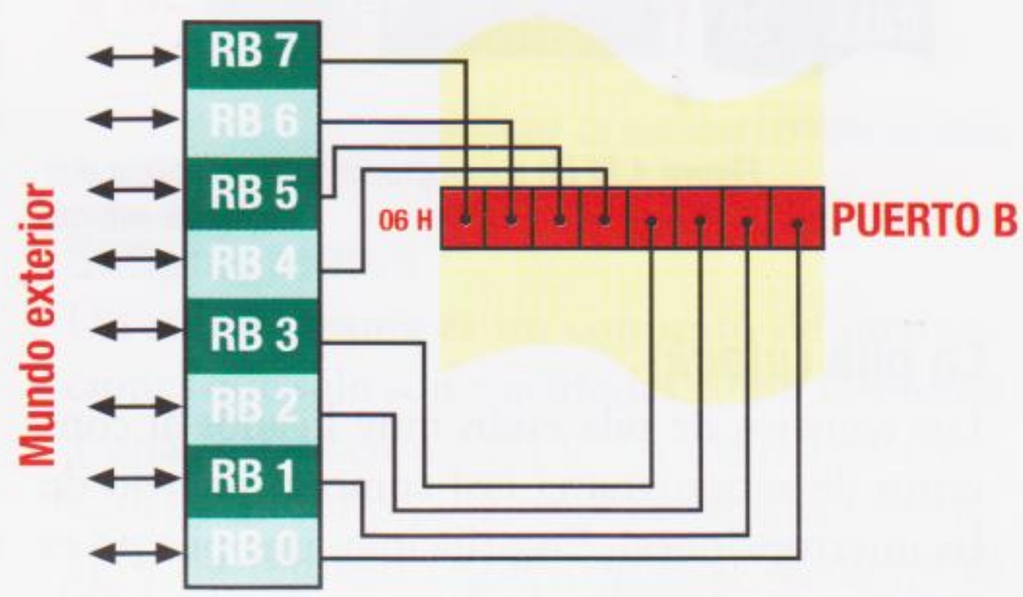


Figura 4.19 *El puerto B*

Configuración de los puertos A y B

Cada *bit* de un puerto puede ser configurado, de manera individual, como entrada o como salida, dependiendo de las necesidades particulares del proyecto.

Un mecanismo simple para configurar los *bits* de los puertos A y B, ya sea como entrada o como salida, consiste en emplear los registros TRISA y TRISB del banco 1 de la memoria de registros en RAM. La **figura 4.20** muestra la relación entre los puertos A y B y los correspondientes registros de programación. Al estudiar el *set* de instrucciones se verán en detalle.

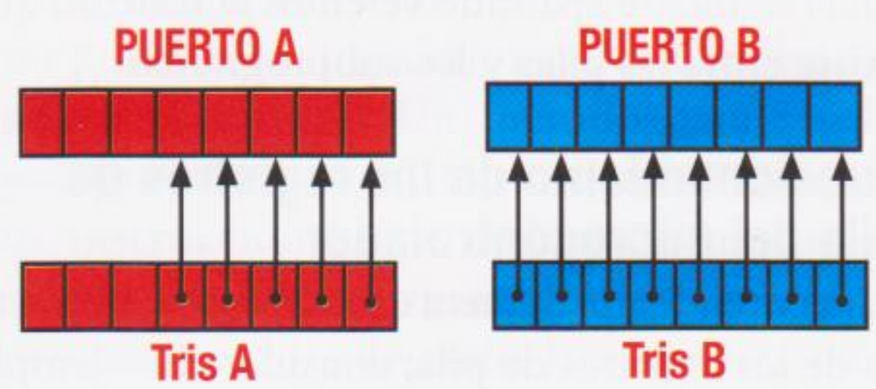


Figura 4.20 *Registros TRISA y TRISB*

LECCIÓN 5

HERRAMIENTAS DE SOFTWARE PARA EL DESARROLLO DE APLICACIONES BASADAS EN MICROCONTROLADORES PIC

La elaboración de un circuito de control embebido requiere el uso de diversas herramientas de *hardware* y de *software*. Estos elementos están conformados por tarjetas electrónicas y programas de soporte, y son distribuidos por un gran número de compañías a lo largo de todo el mundo. Algunas obsequian a los usuarios los programas básicos, como el ensamblador y el grabador de memoria, lo que ha permitido el uso masivo de los microcontroladores.

Elementos orientados al *hardware*

Los elementos de *hardware* se utilizan para verificar el correcto funcionamiento del prototipo construido, así como para almacenar en la memoria el código binario ejecutable. A grandes rasgos, estos componentes son los siguientes:

- **Grabador de memoria o programador**: es un sistema electrónico que permite almacenar el programa en la memoria del microcontrolador. Para conseguir esto, es común que se conecte el grabador al puerto serie o paralelo de una computadora personal, de modo que mediante el uso de un programa especialmente preparado para ello, se hace el envío al microcontrolador del código binario ejecutable. Los grabadores más sofisticados permiten almacenar programas sobre diferentes modelos de microcontrolador, permitiendo elegir a voluntad la referencia deseada. Un ejemplo de un grabador es el K-175 de la compañía CEKIT S.A.

- **Sistema de desarrollo**: es un equipo que combina *hardware* y *software* y permite realizar el ciclo completo en el proceso de elaboración de un prototipo. Al igual que el grabador de memoria, el sistema de desarrollo es controlado por un programa de computadora personal que permite editar, compilar y depurar (corregir) el programa de usuario. Alcanzado este punto, el usuario puede grabar la memoria del microcontrolador con el programa elaborado, así como establecer las conexiones requeridas con los periféricos de entrada-salida dispuestos sobre la tarjeta base. Cuando el prototipo funciona adecuadamente, se procede a la construcción del sistema final sobre una tarjeta independiente. Un ejemplo de un sistema de desarrollo es el **PICSTART Plus** de **Microchip**.

Elementos basados en software

En esta lección nos vamos a concentrar en el estudio de las herramientas de *software* imprescindibles para el desarrollo completo de un proyecto de control con microcontrolador. Estos elementos son:

1. Un editor de texto
2. Un ensamblador
3. Un grabador de memoria
4. Un simulador

El **simulador** es una alternativa opcional que permite probar el funcionamiento de un programa a nivel de *software*, sin que se requiera la construcción física del proyecto. Muchos simuladores incorporan las restantes herramientas de *software*, lo que simplifica en gran medida el desarrollo de los programas de control. Sin embargo, debe quedar claro que, una vez verificado el prototipo con el simulador, éste debe construirse de manera real. Un ejemplo de un simulador es el **MPLAB** de **Microchip**. Este programa contiene todo lo que se necesita para simular el funcionamiento de un prototipo de control.

En la **figura 5.1** se observa el papel que juegan estas herramientas en el ciclo de desarrollo de un proyecto. El orden que debe seguirse es importante: en primer lugar, escribir el programa fuente (**editor de texto**), a continuación generar el código binario ejecutable (**ensamblador**) y por último, programar el microcontrolador (**grabador de memoria**).

Figura 5.1 *El ciclo de desarrollo de un proyecto*

Vamos a describir cada una de las herramientas de *software* indicadas en la **figura 5.1**.

El editor de texto

El primer paso en la elaboración de un programa de microcontrolador consiste en utilizar un editor de texto, similar al bloc de notas de *Win-*

dows o el programa *Edit* del sistema operativo DOS, para escribir el archivo fuente con la solución del problema en lenguaje ensamblador. ***La extensión que debe asignarse al archivo es ASM.***

Para abrir el bloc de notas, sitúese en el escritorio de *Windows* y pulse la secuencia **Inicio**, **Programas**, **Accesorios**, **Bloc de Notas**. Una vez digitado el programa, **figura 5.2**, se lo puede grabar en disco con la opción de menú **Archivo | Guardar como ...**, eligiendo **Todos los archivos** (*.*) en la caja rotulada **Guardar como archivos de tipo:**, e introduciendo a continuación un nombre con extensión **asm** en la caja rotulada **Nombre de archivo:**

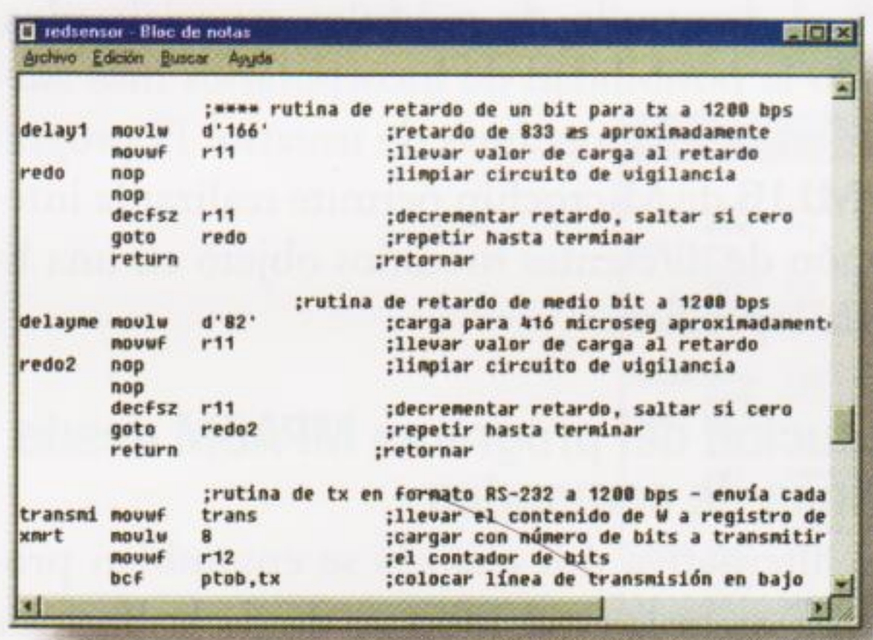


Figura 5.2 *El editor de texto*

El ensamblador

Un programa ensamblador es aquel que permite generar, a partir de un archivo de código fuente, un archivo de código binario.

El archivo fuente debe contener una descripción en lenguaje ensamblador, al nivel del programador, del problema que vamos a resolver. El archivo binario, en contraste, es un conjunto de *bytes* que sólo pueden ser interpretados por el microcontrolador (o microprocesador, cuando se programa en otros niveles). Un ejemplo de un **ensamblador** es el **MPASM** de **Microchip**.

La **figura 5.3** muestra la transformación de un programa fuente (archivo de texto) a su correspondiente programa ejecutable (código binario), orientado al microcontrolador PIC 16F84.

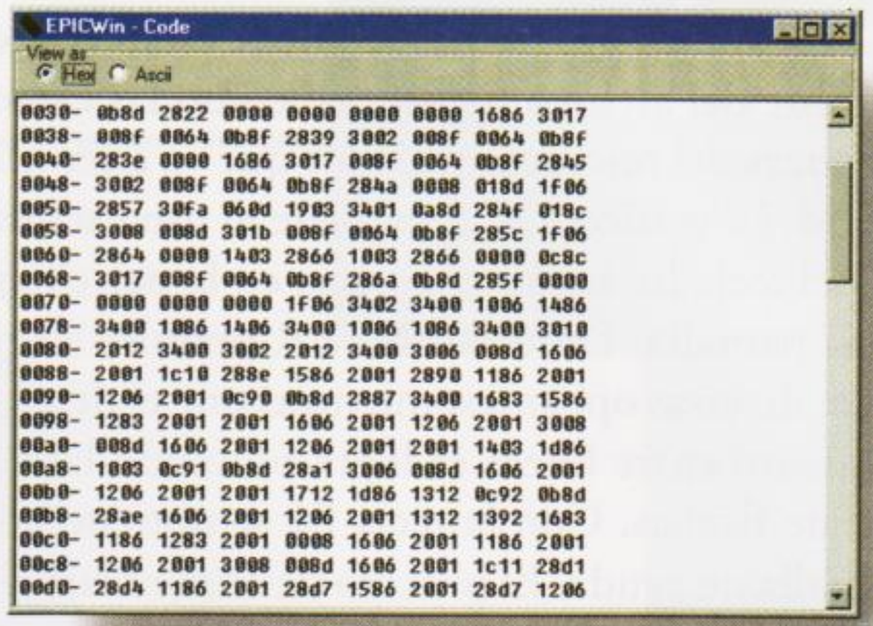


Figura 5.3 *Código binario*

El ensamblador MPASM de Microchip

El ensamblador MPASM de Microchip es un programa que convierte las instrucciones escritas en formato del microcontrolador PIC, a código binario. Se dispone de dos versiones básicas: MPASM para DOS y MPASM para *Windows*. Las **figuras** 5.4 y 5.5 muestran estos programas en ejecución.

Figura 5.4 *MPASM para DOS*

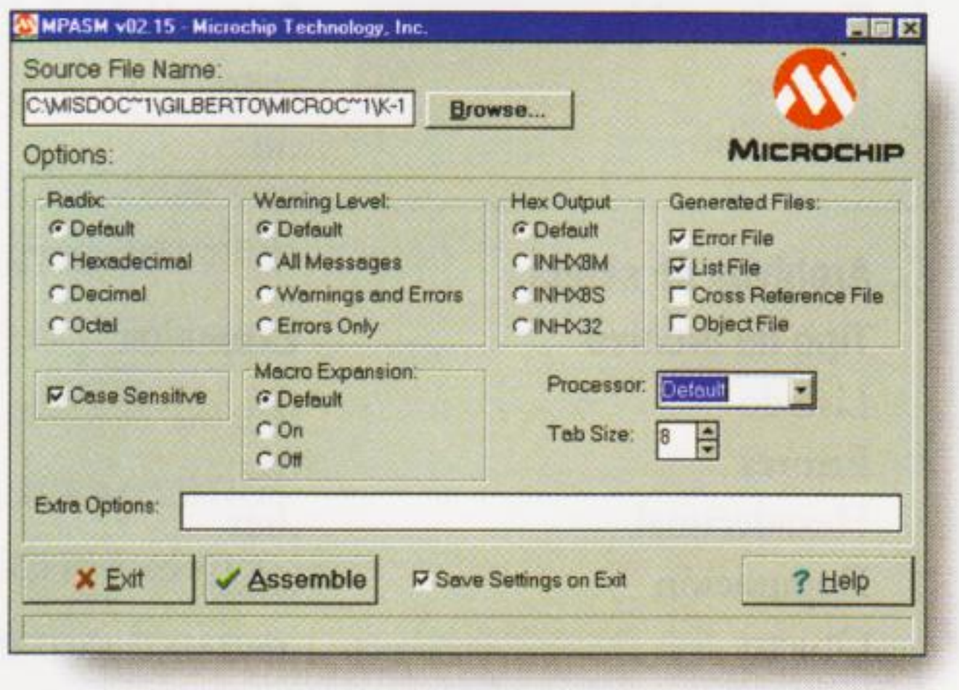


Figura 5.5 *MPASM para Windows*

En ambos programas se puede establecer el nombre del archivo fuente, así como las características del microcontrolador que va a ser utilizado. La versión DOS requiere el uso exclusivo del teclado, según se aprecia en la parte baja de la pantalla. La tecla ENTER permite elegir entre diversas opciones, mientras que el desplazamiento entre líneas se logra pulsando las teclas de flechas. Con la tecla F1 se obtiene una pantalla de ayuda. El proceso de ensamblaje del archivo fuente se consigue pulsando la tecla F10. Para terminar el programa basta con pulsar la tecla ESCAPE.

Debido a su facilidad de uso vamos a emplear en las explicaciones de los próximos apartados la versión MPASM para *Windows*. Las opciones de ensamblaje serán explicadas más adelante.

Archivos asociados al ensamblador MPASM

Para su correcto funcionamiento, el ensamblador MPASM requiere disponer de uno (o varios) archivos de código fuente. El resultado del proceso de ensamblaje es un conjunto de archivos, algunos de los cuales son opcionales y dependen de las necesidades del prototipo que se está desarrollando. La tabla 5.1 muestra los diferentes archivos utilizados y generados por el programa MPASM.

Código absoluto y código objeto

MPASM puede generar dos tipos de archivo ejecutable. El primero de ellos, con extensión **asm**, se conoce como **código absoluto** y puede ser ejecutado directamente por el microcontrolador. El segundo, con extensión **obj**, es un archivo de **código objeto** que puede ser enlazado (empaquetado) con otros módulos similares, dando lugar a programas que combinan diversas funciones diseñadas por separado. El programa que hace la unión (empaquetamiento) de módulos es el **MPLINK** de **Microchip**.

Los archivos con extensión **obj** se pueden agrupar en librerías. Esto es útil, puesto que permite el desarrollo de módulos especializados, dando la posibilidad de incorporarlos más adelante en otros programas de usuario. El programa **MLIB** de **Microchip** permite realizar la integración de diferentes módulos objeto en una librería específica.

Ejecución del programa MPASM desde la línea de comandos

Una alternativa útil cuando se ensamblan programas es realizar el proceso desde la línea de comandos del sistema operativo, no desde el entorno integrado. Esto no parece necesario, dado que los programas del sistema, tanto la versión DOS como la versión *Windows*, cumplen la función básica de ensamblar el código fuente.

Archivos de entrada		
Tipo de archivo	**Extensión**	**Comentario**
Fuente	.asm	Solución del problema al nivel del programador
Inclusión	.inc	Archivos externos que deben ser incluidos
Archivos de salida		
Tipo de archivo	**Extensión**	**Comentario**
Listado	.lst	Salida formateada del proceso de ensamblaje
Errores	.err	Listado de los errores generados
Hexadecimal	.hex	Archivo con el código binario ejecutable
Depuración	.cod	Archivo para depuración
Objeto	.o	Código que se combina con otros módulos

Tabla 5.1 *Archivos de entrada y salida generados por el ensamblador MPASM*

Nota:
Los apartados que siguen son material de referencia para los programas que escribiremos a partir de la próxima lección. Por esta razón, no es obligatorio que usted comprenda todo lo que se expone sobre el uso del ensamblador: cada proyecto subsiguiente aplicará una parte diferente del material teórico expuesto.

El programa ensamblador MPASM puede ejecutarse desde la línea de comandos del sistema operativo DOS con la siguiente sintaxis:

mpasm [/<opcion>[/<opcion>...] [nombre-de-archivo]

Antes de proceder a explicar esta notación, la cual se aplica a muchos tópicos de los microcontroladores, es conveniente ver un ejemplo de su uso:

C:\>proyectos> mpasm /l /e control.asm

Con esta orden se le indica al MPASM que ensamble el archivo **control.asm**, situado en el directorio **C:\>proyectos**. Las opciones usadas en la línea de ordenes son **l** y **e**. La opción **l** le indica al ensamblador que genere un archivo con extensión **lst**, el cual contiene las líneas del programa fuente debidamente formateadas. La opción **e** genera un archivo con extensión **err**. Este archivo contiene una lista con los errores del programa. Si el programa no contiene errores, este archivo estará vacío. Si se producen errores, no se genera el archivo binario ejecutable.

El resultado de la orden mostrada es la creación, dentro del directorio de trabajo, de los siguientes archivos:

Directorio Proyecto		Directorio Proyecto
control.asm—>	MPASM —>	control.asm
		control.cod
		control.hex
		control.lst
		control.err

Archivos generados

Ahora es más fácil explicar la notación usada en la línea de órdenes.

mpasm [/<opcion>[/<opcion>...] [nombre-de-archivo]

La información encerrada entre **corchetes** [ ] es opcional. El símbolo **nombre-de-archivo** hace referencia a cualquier nombre de archivo válido, el cual sirve de base para la creación del archivo binario. En caso de no incluirse, el sistema ejecutará el entorno integrado del MPASM.

El texto que va entre los **paréntesis angulares** < >, es un dato que debe escogerse a partir de un conjunto que ha sido establecido de antemano. En el caso del MPASM, <opción> puede tomar uno de los siguientes valores (aquí no se indican todas las alternativas; en caso de requerirse se deben consultar las ayudas del ensamblador):

Opción	Breve explicación	Parámetro
?	pantalla de ayuda	
a	generación de código absoluto	INHX8M INHX8S INHX32
c	habilita mayúsculas o minúsculas	
e	genera archivo de errores	
l	genera salida formateada	
m	habilita la expansión de macros	
p	define el tipo del procesador seleccionado	
q	permite suprimir la salida a la pantalla	
t	establece el tamaño del tabulador	
r	define el sistema numérico utilizado	HEX(base 16) DEC (base 10) OCT (base 8)

Observe que todas las opciones deben estar precedidas por el carácter **barra inclinada /**. Además, cuando la opción deba elegir uno de entre varios parámetros, éste será colocado a continuación de la opción. Observe los ejemplos:

```
mpasm /pPIC16C54 prueba1.asm   ;procesador=PIC16C54
mpasm /t3 /rHEX prueba2.asm    ;tabulador=3 espacios,
                               ;sistema hexadecimal
```

Los **puntos suspensivos** indican que el elemento que le precede puede repetirse indefinidamente. Si aplicamos este concepto a la línea de órdenes, veremos que se pueden colocar tantas opciones, una después de otra, como queramos.

En resumen, una **opción** en la línea de comandos permite establecer una característica particular del proceso de ensamblaje. Ahora bien, dependiendo del formato utilizado, la opción puede habilitar o inhibir una funcionalidad específica del MPASM.

Formato	Comentario	Ejemplo
/<opción>	Habilita la opción	MPASM /e Genera el archivo de errores
/<opción>+	Habilita la opción	MPASM /e+ Genera el archivo de errores
/<opción>-	Inhibe la opción	MPASM /e- Inhibe el archivo de errores

Directivas

Las directivas son órdenes del ensamblador que no son trasladadas a código binario ejecutable. Ellas se utilizan para controlar diversos aspectos del funcionamiento del ensamblador, como la salida y la entrada de información, el manejo de las macros, al igual que la localización de los datos en la memoria.

Los tipos básicos de directivas incluidas en el MPASM son las siguientes.

- Directivas de control
- Directivas condicionales
- Directivas orientadas a los datos
- Directivas de control de listado
- Directivas de macros
- Directivas orientadas al archivo objeto

Cada programa requiere sus propias directivas, por lo que no es necesario, en un primer acercamiento al tema, considerar todas las posibilidades existentes. Explicaremos aquellas directivas que nos parecen más importantes, dejando para más adelante el estudio de las directivas que se apliquen a los proyectos construidos.

Nota:
Las directivas no son sensibles a las mayúsculas o minúsculas. Esto significa que es lo mismo CONSTANT que Constant.

Presentamos ahora una lista con las directivas clasificadas por categoría. Adicionalmente se brinda una explicación breve y clara de cada directiva, así como la sintaxis que corresponde.

Los proyectos que iniciaremos en la próxima lección harán uso de las directivas mostradas.

Convenciones	
Expr	Hace referencia a una expresión matemática
Valor	Corresponde a un valor literal
Nombre	Nombre asociado a un objeto
Addr	Dirección

Directivas de control

Directiva	Explicación
CONSTANT *sintaxis*	Declara símbolos constantes *constant <etiqueta> [=<expr>,..., <etiqueta> [= <expr>]]*
#DEFINE *sintaxis*	Define una etiqueta usada para sustitución de texto *#define <nombre> [[(<arg>,..., <arg>)]<valor>]*
END *sintaxis*	Finaliza un bloque de programa *end*
EQU *sintaxis*	Declara una constante *<etiqueta> equ <expr>*
#INCLUDE *sintaxis* *sintaxis*	Permite incluir un archivo externo dentro del programa fuente actual *include <<archivo-incluido>>* *include «<archivo-incluido>»*

ORG *sintaxis*	Establece el punto de inicio del programa *<etiqueta> org <expr>*
PROCESSOR *sintaxis*	Define el tipo de procesador utilizado *processor <tipo-de-procesador>*
RADIX *sintaxis*	Especifica el sistema numérico usado por defecto *radix <sistema-por-defecto>*
SET *sintaxis*	Define una variable del ensamblador *<etiqueta> set <expr>*
#UNDEFINE *sintaxis*	Elimina una substitución de etiquetas *#undefine <etiqueta>*
VARIABLE *sintaxis*	Define símbolos variables *variable <etiqueta> [= <expr>,..., <etiqueta> [= <expr>]]*

Directivas condicionales

Directiva	Explicación
ELSE *sintaxis*	Empieza un bloque alternativo a la instrucción IF (si) *else*
ENDIF *sintaxis*	Finaliza un bloque condicional *endif*
ENDW *sintaxis*	Finaliza un ciclo WHILE (mientras) *endw*
IF *sintaxis*	Empieza un bloque de código si la condición es verdadera *if <expr>*
IFDEF *sintaxis*	Ejecuta si el símbolo está definido *ifdef <etiqueta>*
IFNDEF *sintaxis*	Ejecuta si el símbolo no está definido *ifndef <etiqueta>*
WHILE *sintaxis*	Ejecuta un ciclo WHILE (mientras) si la condición es verdadera *while <expr>*

Directivas orientadas a los datos

Directiva	Explicación
_ _BADRAM *sintaxis*	Especifica posiciones de memoria de tipo RAM no válidas *_ _badram <expr>*
CBLOCK *sintaxis*	Define un bloque de constantes *cblock [<expr>]*
_ _CONFIG *sintaxis*	Establece los conmutadores de configuración *_ _config <expr> OR _ _ config <addr>, <expr>*
DA *sintaxis*	Almacena cadenas en la memoria de programa *[<label>] da <expr> [, <expr2>, ..., <exprn>]*
DATA *sintaxis* *sintaxis*	Almacena datos numéricos y de tipo textual *data <expr>,[,<expr>,...]* *data <cadena-de-texto>" [,"<cadena-de-texto>",...]*
DB *sintaxis*	Declara datos cuyo tamaño es un BYTE *db <expr>[,<expr>,...]*
DE *sintaxis*	Declara datos en la EEPROM *de <expr>[,<expr>,...]*
DT *sintaxis*	Define una tabla *dt <expr>[,<expr>,...]*
DW *sintaxis*	Declara datos cuyo tamaño es una PALABRA (WORD) *dw <expr> [,<expr>,...]*

ENDC *sintaxis*	Finaliza un bloque automático de constantes *endc*
FILL *sintaxis*	Especifica un valor para llenar una zona de memoria *fill <expr>, <contador>*

Directivas de control de listado

Directiva *sintaxis*	Explicación *error «<cadena-de-texto>»*
ERROR *sintaxis*	Genera un mensaje de error *error «<cadena-de-texto>»*
ERRORLEVEL *sintaxis*	Establece el nivel de mensajes de error *errorlevel 0|1|2|<+-><mensaje>*
LIST *sintaxis*	Opciones para generar listados *list [<opción>[,...,<opción>]]*
MESSG *sintaxis*	Crea un mensaje definido por el usuario *messg «<mensaje-de-texto>»*
NOLIST *sintaxis*	Inhibe el proceso de listar información *nolist*
PAGE *sintaxis*	Inserta un salto de página *page*
SPACE *sintaxis*	Inserta líneas en blanco *space [<expr>]*
SUBTITLE *sintaxis*	Define un subtítulo para un programa *subtitl «<texto-sub-título>»*
TITLE *sintaxis*	Específica un título para un programa *title «<texto-título>»*

Directivas orientadas a las macros

Estas directivas serán comentadas en la sección de macros.

Directivas orientadas al archivo objeto

Directiva *sintaxis*	Explicación *[<nombre>] code [<addr>]*
CODE *sintaxis*	Inicia una sección de código ejecutable *[<nombre>] code [<addr>]*
EXTERN *sintaxis*	Declara una etiqueta que ha sido definida externamente *extern <etiqueta> [,<etiqueta>]*
GLOBAL *sintaxis*	Exporta a otros módulos etiquetas locales *extern <etiqueta> [.<etiqueta>]*

Macros

Una macro es un conjunto de instrucciones definidas por el usuario, evaluadas y ejecutadas por el ensamblador, cuando ella es llamada desde otro punto dentro del programa.

Una macro se estructura, con más precisión, mediante una secuencia de instrucciones de ensamblador y un conjunto particular de directivas. Las macros pueden aceptar argumentos, lo que las hace extremadamente flexibles, pues se asimilan a procedimientos o funciones de lenguajes de alto nivel. Entre las ventajas de las macros podemos citar:

- Abstracción de procesos
- Código fuente más claro
- Encapsulamiento de procesos que se presentan frecuentemente
- Facilidad de manejo cuando deben realizarse cambios al código
- Verificación simple del funcionamiento de un programa

Una macro es un bloque de código que se escribe *una sola vez*, y que luego puede ser utilizado desde diversos puntos de un programa. Esto simplifica el desarrollo de programas. Una macro puede recibir parámetros, lo que significa que puede particularizar su funcionamiento según las necesidades del proceso que hace el llamado.

Sintaxis de una macro

Una macro se puede definir con la siguiente sintaxis:

<etiqueta> macro [<argumento1>,<argumento2>,...,
<último-argumento>]
:
:
endm

En esta notación, <etiqueta> representa un nombre válido de etiqueta, mientras que <argumento> es cualquier parámetro recibido por la macro. Los valores asignados a estos argumentos en el momento del llamado, son transferidos directamente a los nombres de los argumentos.

El cuerpo de una macro, es decir, el conjunto de elementos que hay en su interior, está conformado por instrucciones del ensamblador y por directivas específicas. El final del cuerpo de la macro se marca con la directiva ENDM.

Directivas orientadas a las macros

Existen directivas específicas para usar dentro de las macros. A continuación se presenta una tabla con las directivas asociadas a las macros, así como una explicación breve y la sintaxis de cada una de ellas.

Directiva	Explicación
ENDM *sintaxis*	Finaliza una macro *endm*
EXITM *sintaxis*	Obliga a salir de la macro *exitm*
EXPAND *sintaxis*	Expande una macro, es decir, la copia en la posición actual *expand*
LOCAL *sintaxis*	Declara una variable local a la macro *local <etiqueta>[,<etiqueta>]*
MACRO *sintaxis*	Permite definir una macro *<etiqueta> macro [<arg>,...<arg>]*
NOEXPAND *sintaxis*	Elimina la expansión de macros *noexpand*

Uso de las macros

Una vez que una macro ha sido definida, se la puede llamar desde cualquier punto de un programa, como se describe a continuación.

<nombre-macro> [<argumento1>,...]

En la sintaxis mostrada, **nombre-macro** es el nombre de una macro definida previamente, y los argumentos son valores que se deben pasar en el momento de la llamada.

La expansión de la macro se realiza directamente en la memoria del microcontrolador, lo que permite, naturalmente, su ejecución. La lista de argumentos de una macro termina simplemente con un espacio en blanco o con un carácter punto y coma (;).

La directiva EXITM brinda un mecanismo flexible para terminar la expansión de una macro.

Mensajes generados por el ensamblador (*messages*)

Los mensajes generados por el ensamblador guardados en un archivo con extensión .err. Si se utiliza la opción /e-, los mensajes no aparecen en la pantalla de la computadora. Si se utiliza la opción /q, conjuntamente con la opción /e-, los mensajes no serán enviados a la pantalla ni grabados en un archivo, aunque si incluidos en la salida.

Los mensajes consisten en un código numérico y un texto explicativo. Algunos mensajes del ensamblador son los siguientes:

Algunos mensajes básicos

301 MESSAGE:
Explicación:
Mensaje del usuario.

303 Program word too large. Truncated to core size.
Explicación :
Este mensaje hace referencia a un uso inadecuado del tamaño de las instrucciones del programa. Por ejemplo, la familia de microcontroladores PIC 16CXXX posee instrucciones de 14 *bits*, por lo que utilizar un formato diferente genera el mensaje indicado.

314 UNKNOWN MESSAGE
Explicación:
Este mensaje se presenta cuando el ensamblador no entiende cuál es el tipo de problema encontrado.

Nota:
Refiérase a las ayudas del programa para una lista completa de mensajes.

Advertencias (*warnings*) detectadas por el ensamblador

Las advertencias son generadas por el ensamblador cuando el código fuente presenta instrucciones o estructuras que pueden reflejar, potencialmente, algún tipo de problema adicional. Para ilustrar lo anterior, presentamos una lista con advertencias típicas.

Algunas advertencias básicas

201 Symbol not previously defined.
Explicación:
El símbolo a que se hace referencia no ha sido definido previamente.

202 Argument out of range. Least significant bits used.
Explicación:
El argumento posee un valor que no puede ser acomodado completamente en el espacio de memoria asignado.

203 Found opcode in column 1.
Explicación:
Se ha encontrado un código de operación en la primera columna de la línea, lo cual es incorrecto, ya que esta columna está asignada a las etiquetas.

208 Label truncated at 32 characters.
Explicación:
La máxima longitud de las etiquetas debe ser de 32 caracteres. Este mensaje indica que la etiqueta actual se ha truncado para cumplir con este requisito.

Nota:
Refiérase a las ayudas del programa para una lista completa de advertencias.

Errores (*errors*) detectados por el ensamblador

Los errores son fallas cometidas por el programador, aunque algunas veces son producto de la ejecución misma del programa, que impiden la generación de código binario.

Algunos errores básicos

101 ERROR:
Explicación:
Mensaje de error generado por el usuario.

102 Out of memory.
Explicación:
No hay espacio en memoria para la expansión de macros, definiciones o procesamiento interno.

103 Symbol table full.
Explicación:
No hay memoria disponible para la tabla de símbolos.

104 Temp file creation error.
Explicación:
No puede crearse un archivo temporal de uso interno. Verifique el espacio disponible en disco.

105 Cannot open file.
Explicación:
No se puede abrir el archivo de disco indicado debido a que el archivo no existe (cuando es archivo de entrada) o el disco está protegido contra escritura (cuando es archivo de salida).

Nota:
Refiérase a las ayudas del programa para una lista completa de errores.

Operadores, operandos y expresiones

Para terminar la exposición de los conceptos teóricos que se deben conocer para acometer la tarea de construir programas, vamos a describir algunos elementos que, pese a su simplicidad, son de gran importancia práctica. Estos elementos son: cadenas, códigos de escape (ANSI'C), sistemas numéricos, los operadores aritméticos y lógicos y, por último, los códigos ASCII.

Cadenas (*strings*)

Una cadena (*string*) es una secuencia de caracteres, pertenecientes al código estándar ASCII, encerrados entre comillas. La longitud de una cadena está comprendida en el rango 0 a 255, lo que significa que la cantidad de caracteres que se pueden almacenar dentro de la cadena no puede sobrepasar estos límites.

La directiva DW almacena una cadena, de manera automática, en posiciones sucesivas de la memoria. Esto es importante porque permite un significativo ahorro de tiempo a la hora de almacenar cadenas en la memoria. Para ilustrar lo anterior, considere la siguiente directiva:

dw "prueba de texto"

Con el uso de esta directiva se almacenan en la memoria los caracteres comprendidos entre las comillas (no se incluyen las comillas en el texto almacenado).

La secuencia de códigos de escape ANSI'C

Los sistemas operativos han popularizado el uso de las secuencias de escape ANSI'C, las cuales consisten en caracteres especiales de control. El ensamblador MPASM acepta este método de representación.

Los códigos de escape ANSI'C los podemos ver a continuación:

Código de Escape	Descripción	Valor hexadecimal
\a	Alarma sonora (*alert*)	07
\b	Retroceso (*backspace*)	08
\f	Salto de página (*form feed*)	0C
\n	Nueva línea (*new line*)	0A
\r	Retorno de carro (*carriage return*)	0D
\t	Tabulación horizontal (*horizontal tab*)	09
\v	Tabulación vertical (*vertical tab*)	0B
\\	Barra invertida (*backslash*)	5C
\?	Interrogación (*question mark*)	3F
\'	Comilla simple (*apostrophe*)	27
\"	Doble comilla (*double quote*)	22
\0OO	Número octal (*octal digit*)	
\xHH	Número hexadecimal (*hexadecimal number*)	

Sistemas numéricos

El ensamblador MPASM reconoce tres sistemas de numeración: hexadecimal (base 16), decimal (base 10) y octal (base 8). A estos sistemas se deben agregar las representaciones ASCII y binaria.

Cuando MPASM se inicia, asume por defecto el sistema hexadecimal. Esto significa que las asignaciones utilizan este sistema de manera automática.

Sistema	Sintaxis	Ejemplo
Decimal	D'<dígitos>'	D'200'
	.<digitos>	.200
Hexadecimal	H'<dígitos>'	H'3f'
	0x<dígitos-hex>	0x3f
Octal	O'<dígitos-oct>'	O'347
Binario	B'<dígitos-bin>'	B'00110110'
ASCII	A'<carácter>'	A'H'
	'<carácter>'	'H'

Operadores numéricos

Los operadores numéricos básicos reconocidos por MPASM son muy abundantes. En esta sec-

ción vamos a mencionar aquellos que puedan presentar algún interés especial:

Operador	Descripción
$	Contador de programa actual
!	Complemento lógico
-	Negación en complemento a dos
~	Complemento
high	Retorna el byte alto de una palabra
low	Retorna el byte bajo de una palabra
upper	Retorna el byte *superior*
%	Módulo (lo que sobra de una división entera)
<<	Desplazamiento a la izquierda
>>	Desplazamiento a la derecha
==	Comparación
=	Asignación
&	AND entre bits
^	OR exclusivo entre bits
|	OR entre bits
&&	AND lógico
||	OR lógico
+=	m += h equivale a m = m + h
-=	m -= h equivale a m = m - h
*=	m *= h equivale a m = m * h
/=	m /= h equivale a m = m / h
++	m++ equivale a m = m + 1
—	m— equivale a m = m - 1

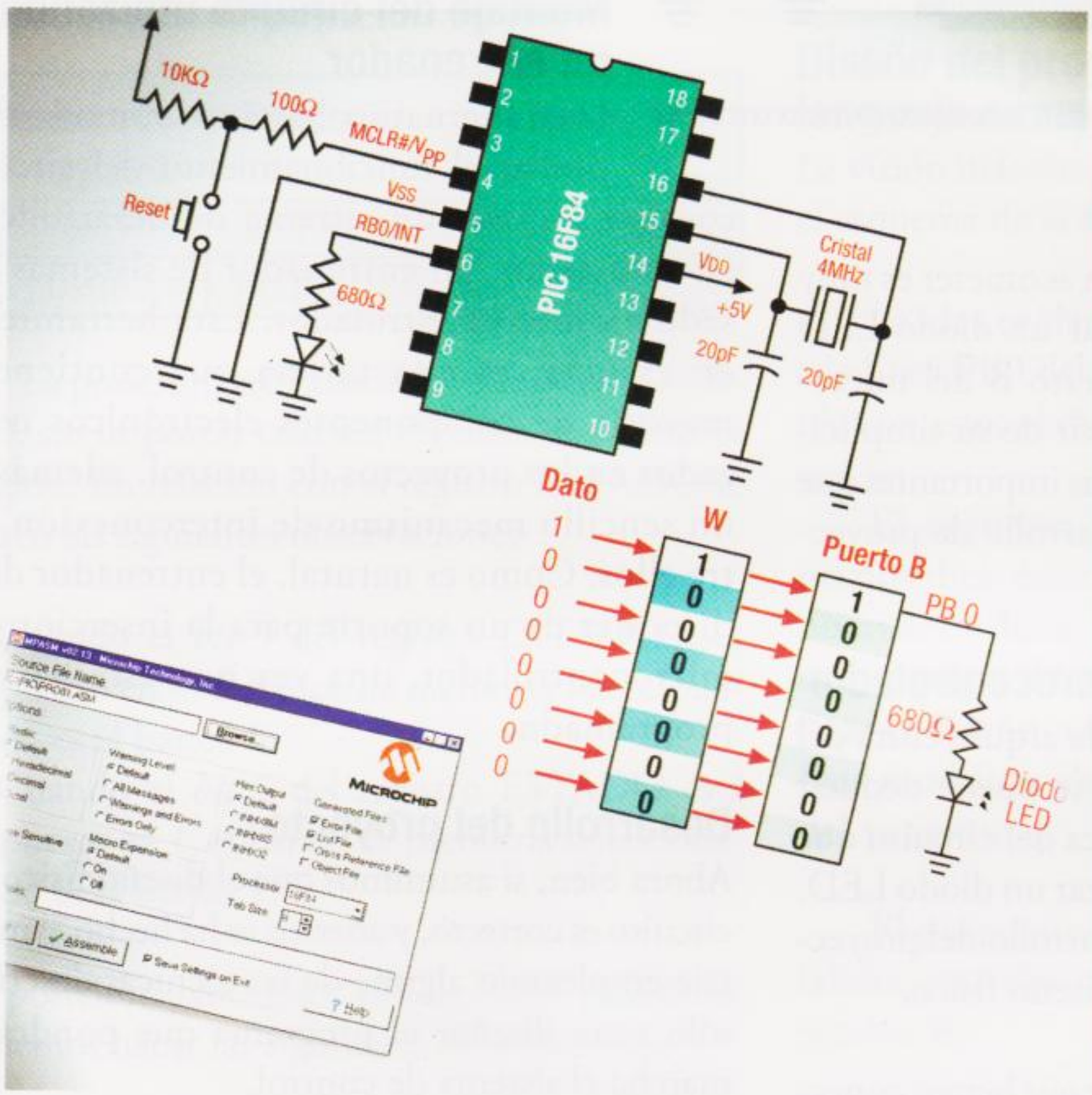
Para ilustrar el uso de la tabla de códigos ASCII, considere el carácter 'A'. Observe que se encuentra dentro de la columna 4 y la fila 1 (0x41).

Carácter	Código ASCII
'A'	0x41

Tabla de códigos ASCII

Hex	0	1	2	3	4	5	6	7
0	NUL	DLE	Space	0	@	P	‘	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	“	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	Bell	ETB	‘	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS		=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

LECCIÓN 6



EJERCICIOS DE PROGRAMACIÓN BÁSICA CON MICROCONTROLADORES PIC

En esta lección vamos a desarrollar varios proyectos de programación en los que aplicaremos algunas de las técnicas aprendidas hasta ahora combinándolas con algunas herramientas, tales como el ensamblador y el grabador de la memoria del microcontrolador.

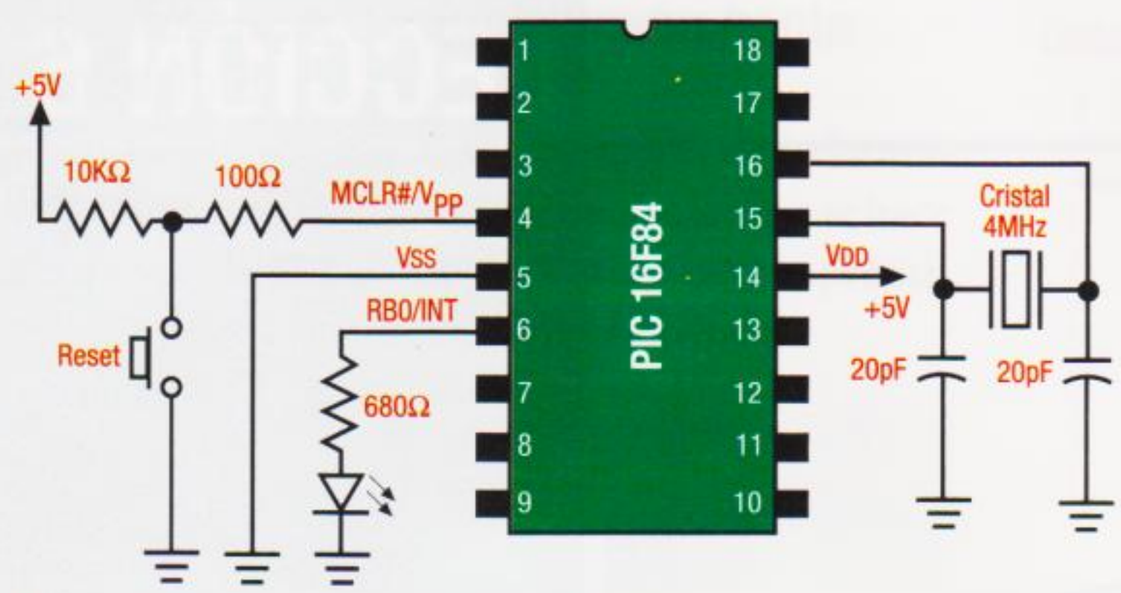


Figura 6.1 *Circuito para energizar un diodo LED, empleando un microcontrolador*

Ejercicio Nº 1

El proyecto inicial que vamos a acometer es muy simple: el objetivo es energizar un diodo LED conectado a una línea del puerto B del microcontrolador PIC16F84. A pesar de su simplicidad, este proyecto contiene ideas importantes que luego van a ser clave en el desarrollo de proyectos de mayor envergadura.

Circuito basado en un microcontrolador

A partir del conocimiento de la arquitectura del microcontrolador PIC16F84 se puede deducir cual debe ser la estructura física del circuito que resuelve el problema de energizar un diodo LED. Este es el paso inicial en el desarrollo del proyecto. La **figura 6.1** muestra el diseño físico.

En este diseño se debe notar que hemos conectado un diodo LED a la línea 0 del puerto B (PB0). Dado que sobre el diodo LED se aplica una tensión de 5 voltios, es necesario que esta línea se comporte como una línea de salida. Esto se consigue, como se verá más adelante, usando técnicas de programación.

Montaje del circuito utilizando un protoboard

Un método convencional utilizado en el desarrollo de circuitos electrónicos, es el armado del circuito en un tablero de conexiones (*protoboard*). Los componentes se insertan sobre él de acuerdo con la disponibilidad de los canales, procediendo después a la interconexión de los mismos con cables del grosor apropiado. La **figura 6.2** presenta el circuito usando un *protoboard*.

El uso de un *protoboard* es una técnica viable cuando el número de elementos utilizados no es muy alto. El inconveniente de su uso tiene que ver con la inestabilidad de las conexiones entre componentes, lo que hace difícil la puesta a punto del circuito.

Montaje del circuito utilizando un entrenador

Una alternativa bastante cómoda para probar el funcionamiento del circuito consiste en usar un sistema de desarrollo o, en su defecto, un entrenador de sistemas basado en microcontrolador. Esta herramienta en esencia, es una tarjeta que contiene la mayoría de componentes electrónicos utilizados en los proyectos de control, además de un sencillo mecanismo de interconexión entre ellos. Como es natural, el entrenador debe disponer de un soporte para la inserción del microcontrolador, una vez que éste ha sido programado.

Desarrollo del proyecto

Ahora bien, si asumimos que el diseño físico del circuito es correcto, y además se ha hecho el montaje empleando alguna de las técnicas descritas, sólo resta diseñar el programa que pondrá en marcha el sistema de control.

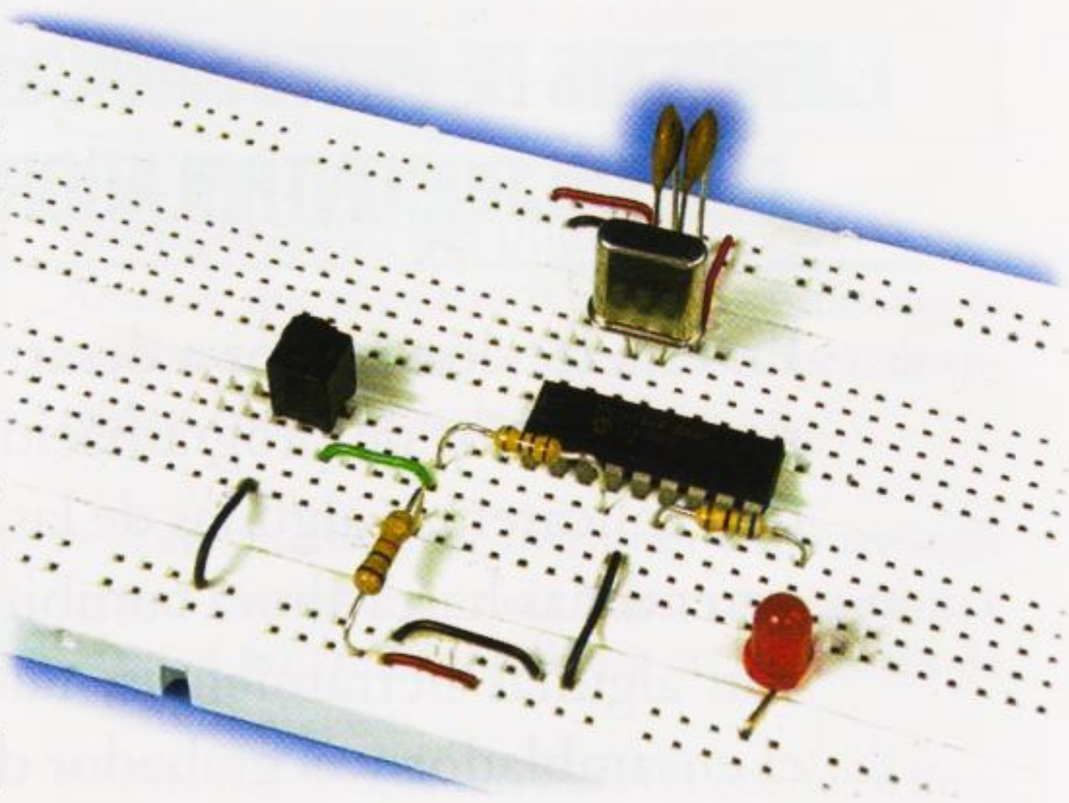


Figura 6.2 *Montaje del circuito sobre un protoboard*

Vamos a proceder en orden. Para empezar, diseñaremos el programa que cumpla con los requisitos del proyecto.

Registros utilizados

Antes de empezar, queremos que usted recuerde el significado de algunos registros del microcontrolador, en particular aquellos que van a ser utilizados en nuestro proyecto.

Número del registro	Banco 0	Banco 1
0x03	ESTADO	ESTADO
0x06	PUERTOB	TRISB

La posición 0x03 contiene el registro ESTADO, sin importar cual sea el banco de memoria elegido. En relación con el registro ESTADO se tienen las siguientes observaciones:

- Cuando el *bit* 5 del registro ESTADO está puesto en 0, el banco de memoria seleccionado es el banco 0.
- Cuando el *bit* 5 del registro ESTADO está puesto en 1, el banco de memoria seleccionado es el banco 1.

La posición 0x06 hace referencia al puerto B. Podemos hacer las siguientes observaciones:

- Cuando el banco seleccionado es el banco 1, el registro 0x06 contiene una imagen de la información contenida en el puerto B.
- Cuando el banco seleccionado es el banco 0, el registro 0x06 contiene la palabra de configuración del puerto B.

Cada *bit* de la palabra de configuración establece el comportamiento de la respectiva línea del puerto del microcontrolador. Notemos lo siguiente:

Figura 6.3 *Objetivo central del proyecto*

- Un *bit* puesto en 1 en el registro de configuración hace que la correspondiente línea del puerto se transforme en una línea de entrada.
- Un *bit* puesto en 0 en el registro de configuración hace que la correspondiente línea del puerto se transforme en una línea de salida.

Diseño del programa usando el lenguaje ensamblador

La visión más simple del proyecto se presenta en el esquema de la **figura 6.3**.

La idea es clara: transferir el dígito binario 1 a la línea PB0 del puerto B. Tan pronto se cumpla con esto, el diodo LED se encenderá.

El esquema presentado en la figura 6.3 no nos dice nada sobre las restantes líneas del puerto, es decir, PB1 a PB7. Esto resulta ser útil, porque de esta manera podemos simplificar el diseño, tal como se ve en la **figura 6.4**. Esta modificación no altera el funcionamiento del programa.

El propósito del proyecto es, en pocas palabras, transferir el dato binario 00000001 al puerto B.

Nota:
El número binario 00000001 se representa más fácilmente con la notación hexadecimal. El equivalente del número binario 00000001 es 0x01 en hexadecimal.

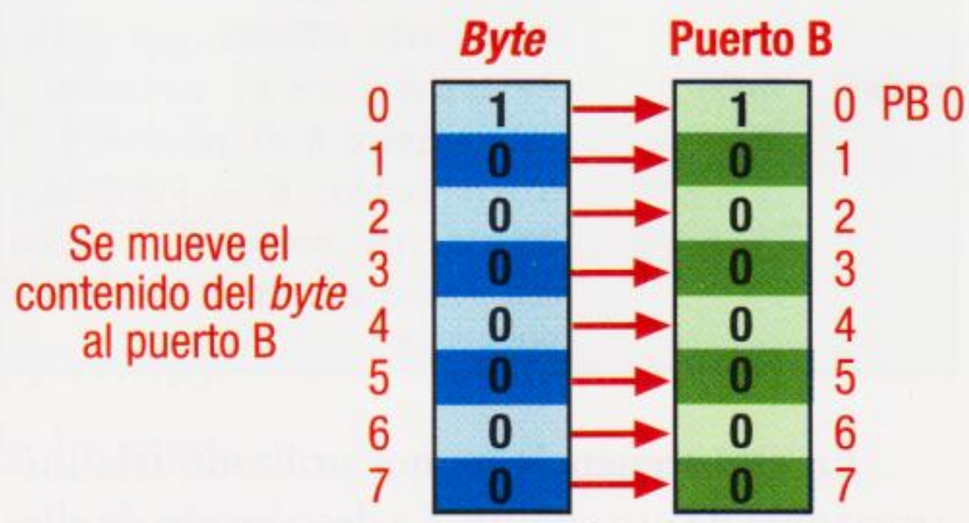


Figura 6.4 *Rediseño del objetivo central del proyecto*

Transferencia de información hacia el puerto B

Debe recordarse que la transferencia de información entre el microcontrolador y su entorno se hace a través de registros. Por esta razón es comprensible que utilicemos la siguiente secuencia:

0x01 ————> Registro W
Registro W —> Puerto B

Lo primero es almacenar en el registro W la información que va a ser transferida (00000001). A continuación, se debe mover el dato en el registro W (00000001) hacia el puerto B. El resultado, si el puerto B es de salida, es la aplicación de un voltaje positivo al diodo LED conectado en la línea PB0 (*bit* 0) del puerto B. La **figura 6.5** presenta un esquema que refuerza estos conceptos.

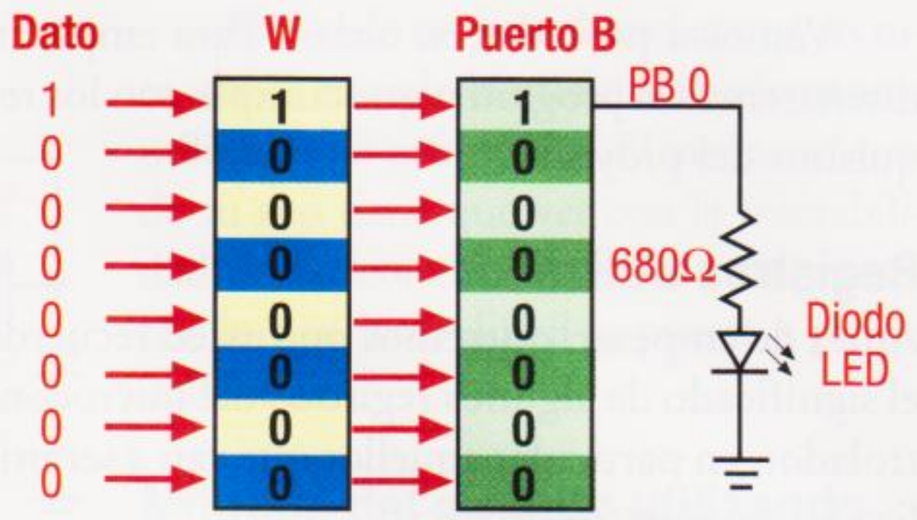


Figura 6.5 *Aplicación de una señal eléctrica al diodo LED*

Almacenamiento del dato 0x01 en el puerto B

De manera más específica el proceso puede enunciarse así:

Seleccionar el banco 0
Almacenar en el registro W el dato 0x01
Mover el contenido del registro W al puerto B

Usando el lenguaje ensamblador se tiene:

```
bcf     0x03,5     ;Coloca 0 en el bit 5 del
                   ;registro ESTADO. Esta acción
                   ;selecciona el banco 0 del
                   ;conjunto de registros
movlw   0x01       ;Se coloca en 1 el bit de menor
                   ;peso del registro W.
                   ;Significado: 00000001 -> W
movwf   0x06       ;Se transfiere el contenido
                   ;del registro W al puerto B.
                   ;Significado: W -> PUERTOB.
                   ;Esta acción energiza el diodo
                   ;LED conectado en el bit 0 del
                   ;PUERTO B
```

En esta secuencia hemos utilizado tres instrucciones de uso común. La descripción de ellas es la siguiente:

bcf registro,dato

La instrucción **bcf** utiliza dos operandos. El primero de ellos es el registro afectado; el segundo corresponde a un bit dentro del registro. Esta instrucción coloca en el *bit* seleccionado el número 0, sin afectar los restantes *bits* del registro.

movlw dato

La instrucción **movlw** tiene un solo operando. Esta instrucción almacena el valor del operando en el registro W. Otra forma de expresarlo es como sigue: **dato** ——> **W**.

movwf registro

La instrucción **movwf** dispone de un solo operando. Este operando hace referencia a un registro. La instrucción **movwf** transfiere el contenido del registro W al registro especificado. En términos simples:

W ——> registro.

Configuración del puerto B

Antes de que el puerto B pueda ser utilizado, es necesario configurar sus líneas como líneas de entrada o como líneas de salida.

- Para transformar una línea del puerto en una línea de entrada basta con escribir 1 en el *bit* correspondiente en la palabra de configuración.
- Para transformar una línea del puerto en línea de salida, basta con escribir 0 en el *bit* correspondiente en la palabra de configuración.

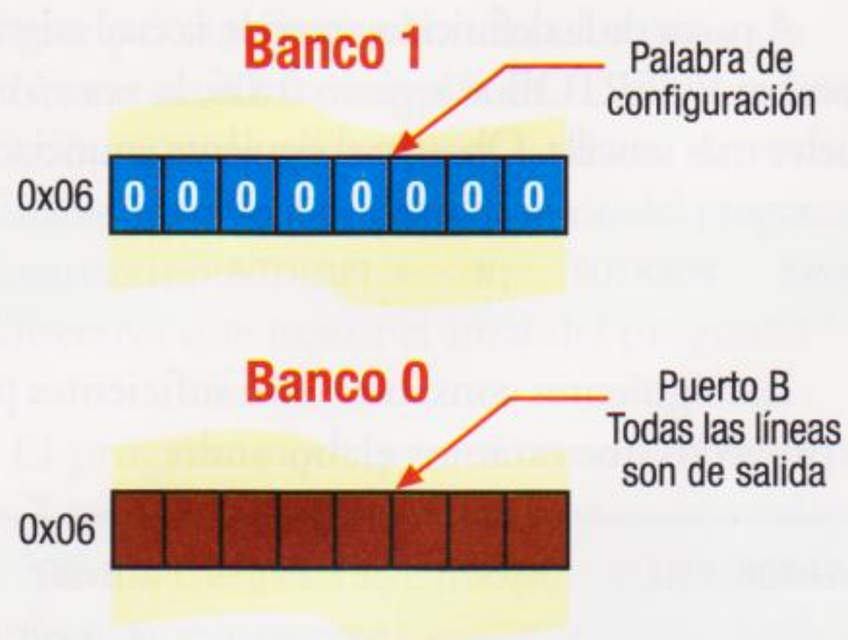


Figura 6.6 *Configuración del puerto B como puerto de salida*

La **figura 6.6** establece lo anterior y muestra como se debe configurar el puerto B para que todas sus líneas sean de salida.

Para configurar el puerto B como puerto de salida se requieren dos pasos:

1. Seleccionar el banco 1
2. Llenar con ceros el contenido del registro de configuración del puerto B

En lenguaje ensamblador tendríamos:

```
bsf     0x03,5          ;Coloca 1 en el bit 5
                        ;del registro de estado.
                        ;Esta acción selecciona
                        ;el banco 1 del conjunto
                        ;de registros
clrf    0x06            ;Almacena ceros en el
                        ;registro 0x06 (TRISB).
                        ;Dado que el banco 1 está
                        ;seleccionado, el puerto
                        ;B se configura como
                        ;puerto de salida
```

Esta secuencia contiene algunas instrucciones nuevas. Vamos a explicarlas brevemente.

bsf registro,número_*bit*

La instrucción **bsf** utiliza dos operandos. El primero corresponde a uno de los registros del microcontrolador. El segundo se refiere a uno de los *bits* del registro. Esta instrucción coloca en 1 el *bit* seleccionado, pero sin afectar los restantes *bits* del registro.

clrf registro

La instrucción **clrf** utiliza un operando que corresponde a un registro del microcontrolador. Esta instrucción coloca ceros en todos los *bits* del registro seleccionado.

Detención del flujo de ejecución

El programa debe terminar de manera controlada. Cuando todas las tareas han sido completadas es conveniente que el microcontrolador entre en un ciclo cerrado durante el cual no ejecuta ninguna tarea. Esto detiene el proceso de ejecución.

La instrucción **goto** (saltar a ...) permite configurar el mencionado ciclo. Vamos a explicar brevemente la instrucción **goto**.

goto etiqueta

La instrucción **goto** utiliza un operando que debe coincidir con una etiqueta definida dentro del programa. Esta instrucción obliga al microcontrolador a continuar el flujo de ejecución a partir de la línea que contiene dicha etiqueta.

Veamos un ejemplo que muestra como crear un ciclo cerrado:

```
ciclo   goto    ciclo   ;La instrucción "goto"
                        ;salta a la etiqueta
                        ;"ciclo". Ahora bien,
                        ;puesto que esta etiqueta
                        ;corresponde a la misma
                        ;instrucción, el programa
                        ;no puede avanzar y se
                        ;detiene el flujo de
                        ;ejecución. El sistema
                        ;termina cuando se retira
                        ;la alimentación del
                        ;circuito.
```

Directivas de control

A continuación explicamos en detalle algunas **directivas** que son necesarias cuando se escribe un programa de microcontrolador.

Directiva para establecer el punto de inicio del programa

Las instrucciones deben almacenarse a partir de cierta dirección de memoria. Ahora bien, es necesario indicar mediante una directiva cuál es esa dirección de memoria.

La directiva ORG establece la posición de memoria a partir de la cual se van a ensamblar las instrucciones escritas a continuación. Observe el siguiente modelo:

```
ORG     0   ;Esta directiva establece la
            ;posición a partir de la cual se
            ;ensamblan las próximas instrucciones.
            ;En este ejemplo, la dirección de
            ;ensamblaje es cero.
```

Cuando el programa no utiliza interrupciones puede usar todo el espacio de memoria para almacenar el código binario. En este caso, el programa se graba a partir de la posición 0x00 utilizando la memoria excedente en forma secuencial.

Nota:

Es posible colocar varias directivas ORG a lo largo del programa. De esta manera se pueden almacenar distintos bloques de programa en áreas no contiguas de memoria.

Directiva para definir constantes

En párrafos anteriores utilizamos la siguiente instrucción:

```
movwf   0x06            ; W ---> 0x06
```

Esta instrucción mueve el contenido de W al registro 0x06. Si usted tiene buena memoria recordará, sin duda, que el número 0x06 se refiere al puerto B. Sin embargo es mejor evitar este esfuerzo, y para lograrlo se recurre al uso de la directiva EQU.

Esta directiva permite definir constantes de propósito general. Veamos un ejemplo:

```
PUERTOB EQU 0x06  ;La posición 0x06 corresponde
                  ;al puerto B
```

A partir de la definición anterior, la cual asigna el nombre PUERTOB al registro 0x06, la notación se vuelve más sencilla. Observe el siguiente enunciado:

```
movwf   PUERTOB   ;W --> PUERTOB
```

Las siguientes constantes son suficientes para el proyecto que estamos elaborando:

```
PUERTOB EQU     0x06    ;El registro 0x06
                        ;corresponde al puerto B,
                        ;si el banco 0 está
                        ;seleccionado
TRISB   EQU     0x06    ;El registro 0x06
                        ;permite configurar el
                        ;puerto B, como entrada
                        ;o como salida,
                        ;si el banco 1 está
                        ;seleccionado
ESTADO  EQU     0x03    ;El registro 0x03
                        ;corresponde al registro
                        ;ESTADO, sin importar el
                        ;banco seleccionado
```

Directivas iniciales

El proceso de ensamblaje del código fuente requiere que se especifiquen ciertos elementos que van a incidir en el código generado. El primero de ellos hace referencia al tipo de microcontrolador utilizado. El segundo tiene que ver con el sistema numérico empleado en las diferentes operaciones. La directiva LIST sirve para indicar la referencia del microcontrolador utilizado en el proyecto. Con la directiva RADIX se elige el sistema numérico. Veamos un ejemplo:

```
LIST    p=16F84   ;Se utiliza el PIC 16F84
RADIX   HEX       ;Se utiliza el sistema
                  ;hexadecimal
```

Directiva final

La etapa de ensamblaje termina con la directiva END.

Código fuente del proyecto

Ya estamos preparados para observar el aspecto final del código fuente. La estructura general del programa debe ajustarse al siguiente orden:

- Descripción del microcontrolador utilizado
- Sistema numérico empleado
- Definición de constantes
- Establecimiento del punto de inicio del programa
- Instrucciones
- Directiva que marca el final del programa

El programa que presentamos en la **figura 6.7** resulta de unir las secciones de código que hemos descrito en los párrafos anteriores, de acuerdo con el orden que acabamos de plantear.

```
;=========================================
; PROGRAMA..:      Pro01.asm
; AUTOR.....:      CEKIT S.A.
; FECHA.....:      Julio de 2001
; COMENTARIO:      El programa Pro01.asm
                   ;energiza un diodo LED.
                   ;El programa es ensamblado,
                   ;en secuencia, a
                   ;partir de la dirección cero
;-----------------------------------------
; ENCABEZADO
;-----------------------------------------
        LIST    p=16F84 ;Se utiliza el PIC 16F84
        RADIX   HEX     ;Se utiliza el sistema
                        ;hexadecimal

;-----------------------------------------
; ZONA DE ETIQUETAS
;-----------------------------------
PUERTOB EQU     0x06    ;El registro 0x06
                        ;corresponde al puerto B,
                        ;si el banco 0 está
                        ;seleccionado
TRISB   EQU     0x06    ;El registro 0x06 permite
                        ;configurar el puerto B,
                        ;como entrada o como
                        ;salida, si el banco 1
                        ;está seleccionado
ESTADO  EQU     0x03    ;El registro 0x03
                        ;corresponde al registro
                        ;ESTADO, sin importar el
                        ;banco seleccionado

;-----------------------------------------
; INICIO DEL PROGRAMA
;-----------------------------------------
        ORG     0       ;Esta directiva define
                        ;la posición a partir
                        ;de la cual se
                        ;ensamblan las
                        ;próximas instrucciones.
                        ;En este ejemplo, la
                        ;dirección de ensamblaje
                        ;es cero.
;-----------------------------------------
; CÓDIGO DEL PROGRAMA
;-----------------------------------------
inicio  bsf   ESTADO,5  ;Coloca 1 en el bit 5
                        ;del registro de estado.
                        ;Esta acción selecciona
                        ;el banco 1 del conjunto
                        ;de registros
        clrf  TRISB     ;Almacena ceros en el
                        ;registro 0x06 (TRISB).
                        ;Dado que el banco 1 está
                        ;seleccionado, el puerto
                        ;B se configura 6.como
                        ;puerto de salida
        bcf   ESTADO,5  ;Coloca 0 en el bit 5
                        ;del registro ESTADO.
                        ;Esta acción selecciona
                        ;el banco 0 del conjunto
                        ;de registros
        movlw 0x01      ;Se coloca en 1 el bit
                        ;de menor peso del
                        ;registro W.
                        ;Significado: 00000001
                        ;-> W
        movwf PUERTOB   ;Se transfiere el
                        ;contenido del registro
                        ;W al puerto B.
                        ;Significado: W-> PUERTOB.
                        ;Esta acción energiza
                        ;el diodo LED conectado
                        ;en el bit 0 del PUERTO B
ciclo   goto  ciclo     ;La instrucción "goto"
                        ;salta a la etiqueta
                        ;"ciclo". Ahora bien,
                        ;puesto que esta etiqueta
                        ;corresponde a la misma
                        ;instrucción, el programa
                        ;no puede avanzar y se
                        ;detiene el flujo de
                        ;ejecución. El sistema
                        ;termina cuando se retira
                        ;la alimentación del
                        ;circuito.
;-----------------------------------------
; FINAL DEL PROGRAMA
;-----------------------------------------
        END             ;Termina el proceso de
                        ;ensamblaje
```

Figura 6.7 Código fuente del programa

Escriba este programa usando un editor de texto convencional, como el EDIT del sistema DOS o el **Bloc de notas** de Windows. Guarde el programa con el nombre:

```
Pro01.asm
```

Nota:
No olvide que la extensión del programa debe ser **ASM**. Esto es importante, sobre todo si utiliza el Bloc de notas. En este caso debe elegir en **Guardar como archivos de tipo:**, la opción **Todos los archivos (*.*)**.

Ensamblando el programa

Una vez escrito el programa, es necesario ensamblarlo para generar el código binario correspondiente (archivo con extensión HEX). Vamos a utilizar la versión MPASM para Windows de Microchip. La **figura 6.8** muestra la interfaz visual del ensamblador.

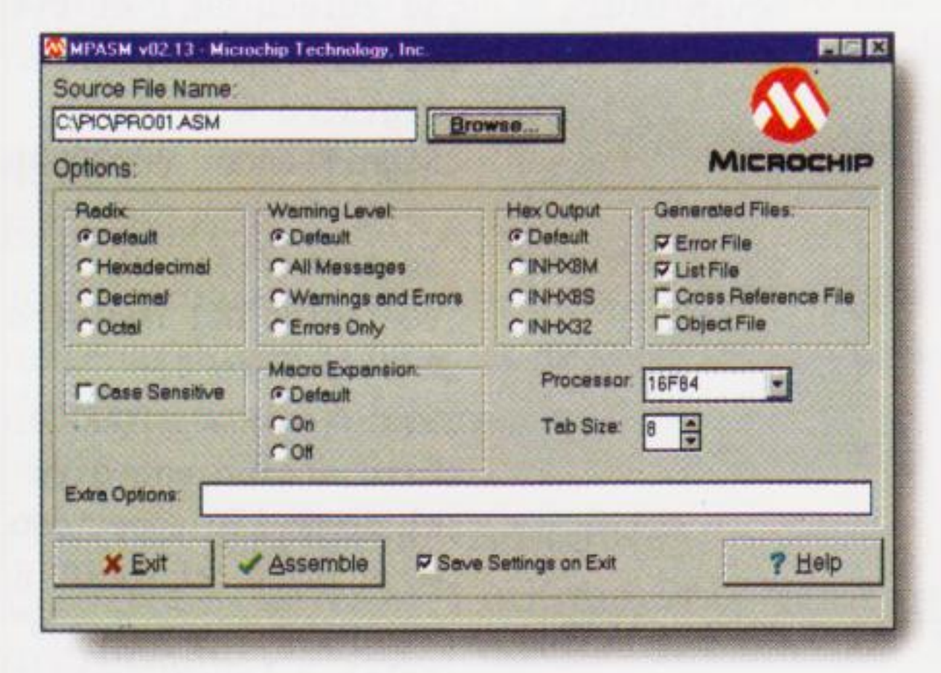


Figura 6.8 El ensamblador MPASM (Windows)

El botón **Browse ...** permite examinar la estructura de directorios del sistema. Observe la **figura 6.9**.

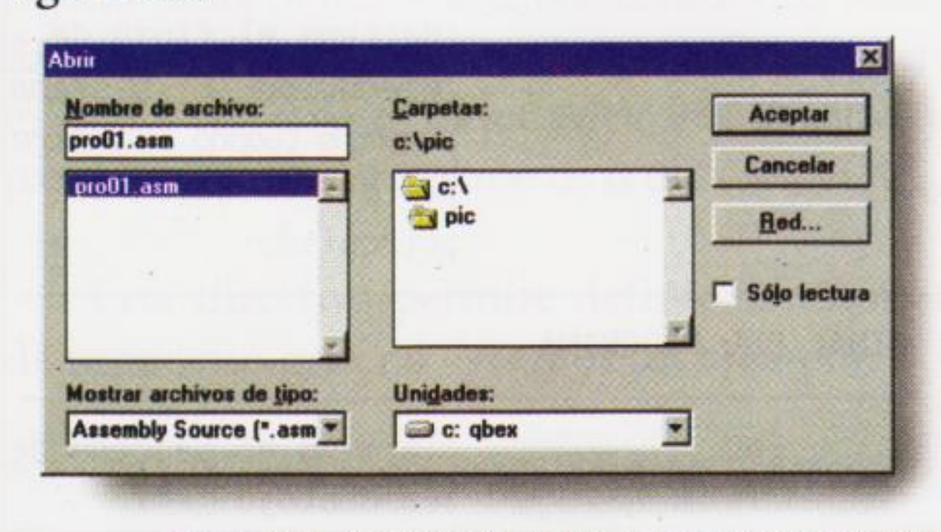


Figura 6.9 Elección del archivo que se va a ensamblar

Una vez elegidas las opciones presentadas por el ensamblador, **figura 6.8**, pulse el botón ***Assemble*** (ensamblar). Pasado un breve tiempo, el ensamblador despliega una caja de diálogo con el estado final de la compilación. Si no ha cometido errores al digitar el programa, obtendrá la pantalla que se ve en la **figura 6.10**.

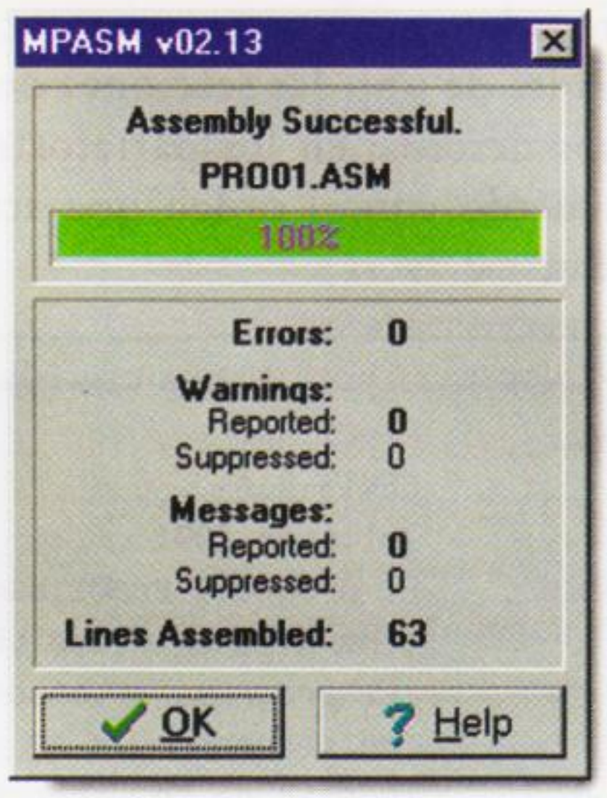


Figura 6.10 Proceso de ensamblaje correcto

Los datos que seguramente aparecerán en su pantalla podrán no coincidir con los mostrados en la **figura 6.10**. Esto se explica porque el número de líneas de su programa podría contener comentarios o líneas en blanco que no aparecen en la **figura 6.7**.

Una vez ensamblado el programa, se originará un conjunto de archivos entre los que se destacan el hexadecimal (pro01.hex) y el del listado (pro01.lst). El archivo del listado nos proporciona información muy útil para localizar posibles errores de escritura o sintaxis que se hayan cometido en el código del programa. El archivo hexadecimal representa la información que se almacenará en la memoria del microcontrolador mediante una tarjeta programadora. En el comercio se pueden conseguir muchas marcas de programadores de gran calidad, entre ellas la tarjeta K-224 de CEKIT, cuya estructura, funcionamiento y manejo se ha estudiado en el proyecto 9 de este curso. Una vez programado el microcontrolador, se debe montar en el *protoboard* o en el circircuito impreso para hacer las pruebas finales y la depuración del sistema final.

Una vez elegidas las opciones presentadas por el ensamblador, **figura 6.10**, pulse el botón *Assemble* (ensamblar). Pasado un breve tiempo, el ensamblador despliega una caja de diálogo con el estado final de la compilación. Si no ha cometido errores al digitar el programa, obtendrá la pantalla que se ve en la **figura 6.12**.

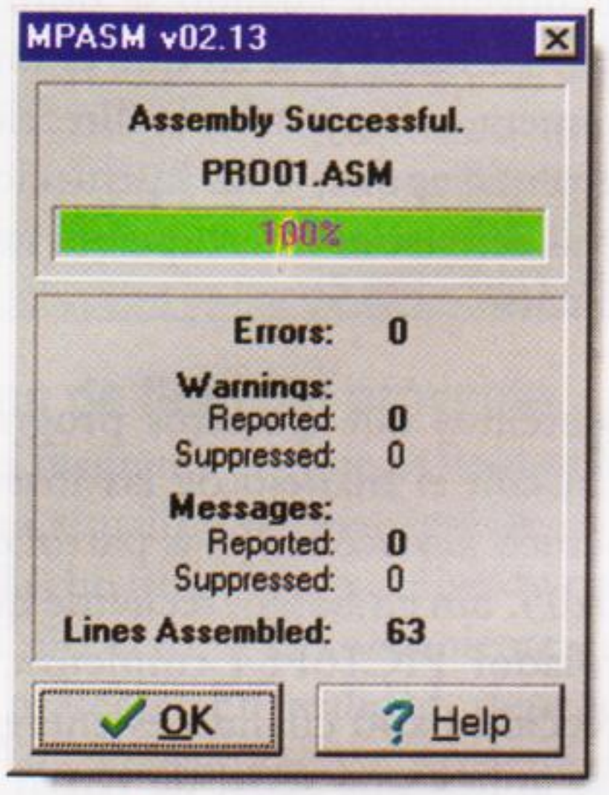


Figura 6.12. *Proceso de ensamblaje correcto*

Los datos que seguramente aparecerán en su pantalla podrán no coincidir con los mostrados en la **figura 6.12**. Esto se explica porque el número de líneas de su programa podría contener comentarios o líneas en blanco que no aparecen en la **figura 6.9**.

Haga clic para cerrar la caja de diálogo. Ahora usted dispone en su directorio de trabajo de los archivos generados por el ensamblador. Observe la lista mostrada en la **figura 6.13**.

Figura 6.13. *Archivos generados en el proceso de ensamblaje*

El siguiente paso consiste en transferir el archivo binario **Pro01.hex** a la memoria del microcontrolador.

Grabando el programa en el microcontrolador

Para grabar un archivo binario en la memoria del microcontrolador usted debe disponer de un dispositivo que cumpla con esta función. Un buen ejemplo de ello es el **Programador PIC K-224** de CEKIT S.A. La **figura 6.14** muestra el Programador K-224.

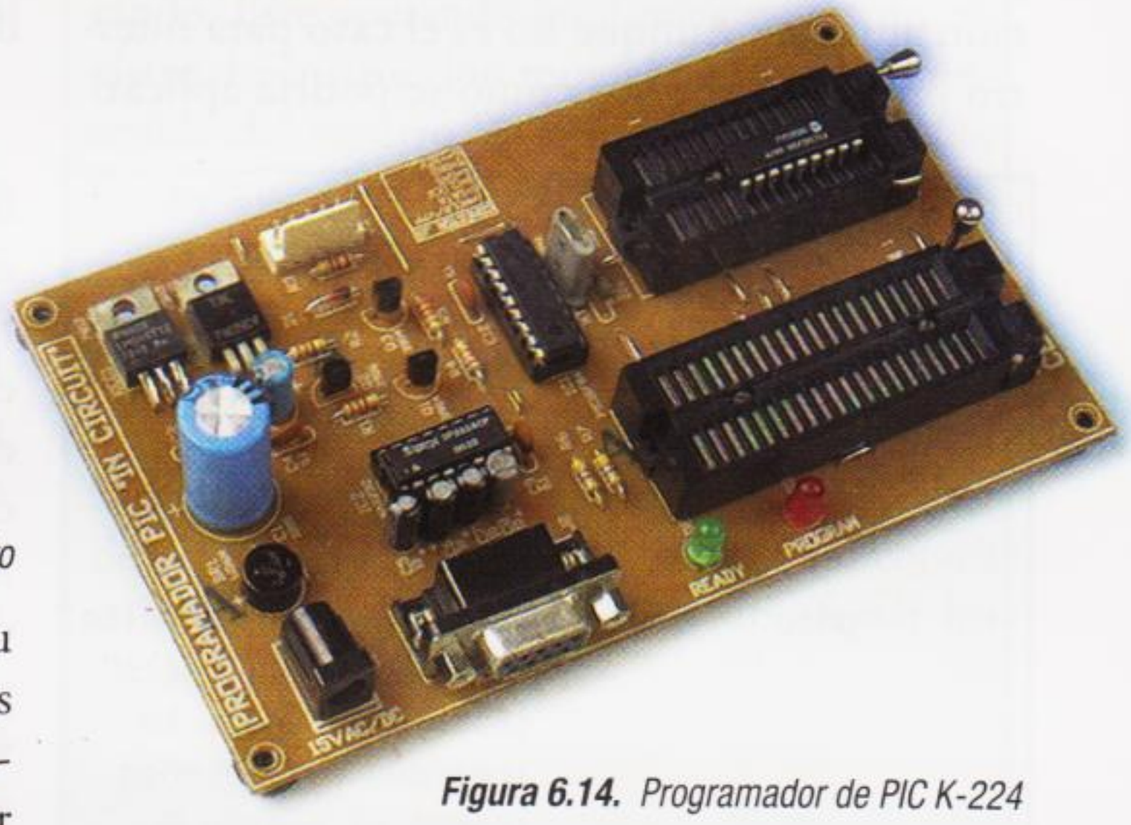


Figura 6.14. *Programador de PIC K-224*

El K-224 debe conectarse al puerto serial de una computadora personal. Esto significa que el proceso de grabación será realizado a partir de un programa de computadora. El manejo de la tarjeta y del programa ya se documentaron ampliamente en el proyecto 9 de esta obra, de manera que proceda a grabar el archivo hexadecimal dentro de la memoria interna del microcontrolador. Los parámetros adecuados de configuración son: Oscilador XT, Protección de código deshabilitado, Watchdog Timer deshabilitado y Power-up Timer deshabilitado. Inserte ahora el microcontrolador en alguno de los sistemas que usted ha elegido para probar su proyecto: el *protoboard*, el entrenador o un circuito impreso. Tan pronto energice el circuito, debería observar como se activa el diodo LED. Si ha tenido éxito, felicitaciones. En caso contrario, no se desanime: sin duda ha omitido algún paso. Repita nuevamente el proceso.

Algunas observaciones finales

En los próximos apartados le mostraremos algunas variantes que usted puede probar en su proyecto y que tienen cierta importancia práctica.

Variantes en el control de terminación del programa

El mecanismo de terminación que le presentamos en la figura 6.9 es el más indicado para el proyecto que estamos desarrollando. En muchas ocasiones, sin embargo, es necesario regresar, dentro del ciclo cerrado, a una instrucción anterior. Aunque no es el caso para nuestro proyecto, observe como se podría aplicar.

```
ciclo   movwf   PUERTOB ;Se transfiere el
                        ;contenido del
                        ;registro W
                        ;al puerto B, es decir,
                        ;W -> PUERTOB. Con esto
                        ;se energiza el led
                        ;conectado al bit 0
                        ;del PUERTO B
        goto    ciclo   ;Esta instrucción salta
                        ;a la etiqueta "ciclo".
                        ;Esto crea un ciclo
                        ;cerrado escribiendo
                        ;sobre el puerto B.
```

El funcionamiento del circuito no se afecta en absoluto. Naturalmente, dependiendo de la línea en donde se coloque la etiqueta se producirán resultados diferentes, lo que tiene que ver con la estructura del programa fuente.

Nota:

Es posible escribir el programa sin emplear un ciclo cerrado al finalizar el código. Observe el código fuente que sigue:

```
movwf   PUERTOB     ;W -> PUERTOB
;-----------------------------------
        END         ;Termina el proceso
                    ;de ensamblaje
```

Este programa funciona igualmente bien, pero no lo recomendamos como técnica válida de programación. Utilice siempre un ciclo cerrado para finalizar el programa.

Haciendo espacio al vector de interrupciones

Un programa empieza siempre a partir de la dirección 0. Por este motivo parece razonable grabar el programa en secuencia, una instrucción después de otra, a partir de la dirección cero. Sin embargo, para el caso del PIC16F84, esto resulta inconveniente cuando se utilizan interrupciones. La razón está en que el microcontrolador espera encontrar en la dirección 0x04 un número que apunte a las instrucciones que manejan las diferentes interrupciones que pueden producirse.

Si queremos que nuestros programas no interfieran con el manejo de las interrupciones, debemos almacenarlos a partir de la dirección 0x05. Sin embargo, recuerde que el microcontrolador PIC16F84 comienza siempre en la dirección 0x00 cuando es energizado (o cuando se aplica una señal de *reset*). Por este motivo es obligatorio colocar una instrucción de salto que nos lleve directamente desde la posición 0x00 hasta la posición 0x05. Observe los cambios que deben hacerse al programa para lograr este objetivo.

```
;-----------------------------------------
; INICIO DEL PROGRAMA
;-----------------------------------------
        ORG     0       ;Esta directiva define
                        ;la dirección a partir
                        ;de la cual se ensamblan
                        ;las próximas instrucciones.
                        ;En este ejemplo la
                        ;dirección es cero.
        goto    inicio  ;Salta a la etiqueta
                        ;"inicio". Esta instrucción
                        ;queda almacenada en la
                        ;dirección cero y se
                        ;activará cuando se
                        ;energice el
                        ;microcontrolador, o
                        ;cuando se produzca una
                        ;señal de reset
;-----------------------------------------
; CODIGO DEL PROGRAMA
;-----------------------------------------
        ORG     5       ;Las siguientes
```

```
                        ;instrucciones se
                        ;ensamblan a partir de
                        ;la dirección 0x05. Si
                        ;a la próxima instrucción
                        ;le asignamos la etiqueta
                        ;"inicio", la instrucción
                        ;de salto "goto inicio"
                        ;(almacenada en 0x00)
                        ;nos llevará directamente
                        ;hasta la dirección 0x05.
                        ;A partir de este punto
                        ;continuará la ejecución
                        ;del programa
inicio  bsf ESTADO,5    ;Coloque un 1 en el bit
                        ;5 del registro ESTADO
```

Diagrama de flujo del proyecto

Aunque el diagrama de flujo es básicamente una ayuda para visualizar la estructura de un programa, y de hecho no lo hemos utilizado en el desarrollo del proyecto, sí es conveniente que usted lo tenga presente. En realidad, depende de la metodología usada; algunos programadores lo usan extensamente y nosotros también lo haremos, pero cuando el proyecto así lo justifique.

La **figura 6.15** muestra el diagrama de flujo que se acomoda al proyecto que acabamos de concluir.

Figura 6.15 *Diagrama de flujo*

Instrucciones utilizadas en esta lección

Para que sirva como referencia presentamos un resumen de las instrucciones utilizadas en esta lección.

bcf registro,dato

La instrucción **bcf** utiliza dos operandos. El primero de ellos es el registro afectado; el segundo corresponde a un *bit* dentro del registro. Esta instrucción coloca en el *bit* seleccionado el número 0, sin afectar los restantes *bits* del registro.

movlw dato

La instrucción **movlw** tiene un solo operando. Esta instrucción almacena el valor del operando en el registro W. Otra forma de expresarlo es como sigue: **dato** ——> W.

movwf registro

La instrucción **movwf** dispone de un solo operando. Este operando hace referencia a un registro. La instrucción **movwf** transfiere el contenido del registro W al registro especificado. En términos simples, W ——> **registro**.

bsf registro,número_*bit*

La instrucción bsf utiliza dos operandos. El primero corresponde a uno de los registros del microcontrolador. El segundo se refiere a uno de los *bits* del registro. Esta instrucción coloca en 1 el *bit* seleccionado, pero sin afectar los restantes *bits* del registro.

clrf registro

La instrucción **clrf** utiliza un operando que corresponde a un registro del microcontrolador. Esta instrucción coloca ceros en todos los *bits* del registro seleccionado.

goto etiqueta

La instrucción **goto** utiliza un operando que debe coincidir con una etiqueta definida dentro del programa. Esta instrucción obliga al microcontrolador a continuar el flujo de ejecución a partir de la línea que contiene dicha etiqueta.

Ejercicio No. 2. Encender y apagar un diodo LED

Propósito general

El objetivo del siguiente ejerccicio es continuar con las ideas básicas desarrolladas en la lección anterior. Vamos a aprovechar el circuito anterior, así como los elementos de programación del

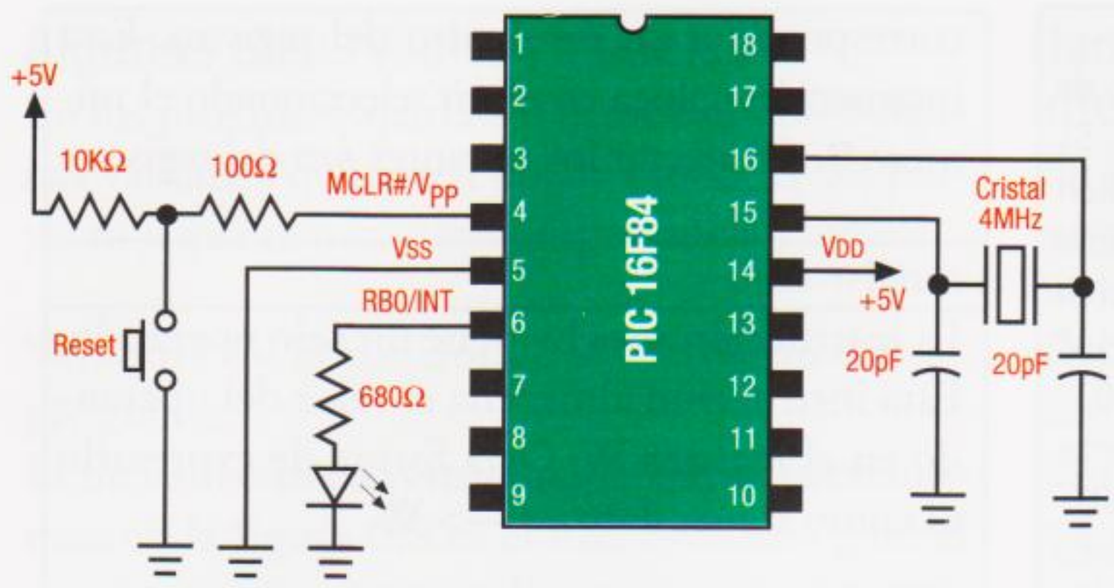


Figura 6.16 *Diseño físico del circuito de control de un diodo LED*

PIC16F84. Como verá, una parte importante del ejercicio anterior tendrá una aplicación completa en el presente ejercicio.

Para empezar, recordemos la estructura del circuito mostrado en la **figura 6.16**.

Propósito específico

El proósito específico de este ejercicioes hacer titilar un diodo LED con una frecuencia constante. Esto se logra aplicando al diodo un voltaje en forma de onda cuadrada, **figura 6.17**.

Análisis

Lo que se pretende en este proyecto es generar una señal de nivel alto en la puerta PB0 durante un cierto intervalo de tiempo y luego una señal de nivel bajo durante el mismo periodo de tiempo. Para que el ojo humano pueda visualizar el cambio de estado en el LED, debemos hacer que cada nivel de salida permanezca durante un intervalo de tiempo; si no lo hacemos, el microcontrolador ejecutará tan rápido el proceso que no veremos ningun cambio.

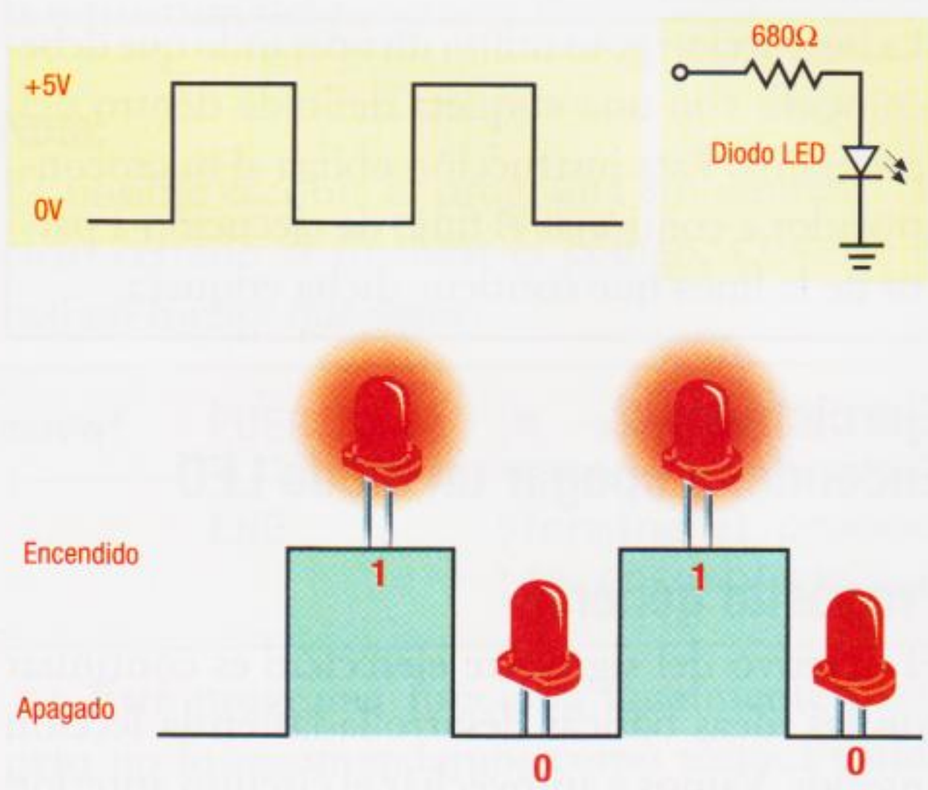


Figura 6.17 *Aplicación de una señal cuadrada a un diodo LED*

A este intervalo de tiempo se le conoce con el nombre de ***retardo***. Pasado este tiempo, el cual se puede configurar usando técnicas de programación, la señal aplicada debe pasar a cero. Ahora bien, el tiempo durante el cual el voltaje permanece en cero también puede ser controlado con instrucciones de programa.

Nota:
Por comodidad, ambos tiempos pueden ser exactamente iguales.

En este punto, si queremos que el proceso se desarrolle de manera repetitiva, basta con ejecutar nuevamente el proceso que se describió en el párrafo anterior. La **figura 6.18** muestra el diagrama de flujo que se acomoda al proceso descrito.

Figura 6.18 *Diagrama de flujo del programa que permite encender y apagar un LED*

Estructura repetitiva

El diagrama de flujo presentado en la figura 6.18 permite ilustrar un aspecto de gran importancia que se presenta al programar un microcontrolador. Este aspecto se refiere a las estructuras cíclicas controladas mediante el uso de una condición lógica. Algunas estructuras repetitivas son simples, como la mostrada en la figura 6.18, pero otras presentan un mayor grado de complejidad. La **figura 6.19** presenta el esquema general de una estructura repetitiva, pero ilustrando un caso particular: una instrucción simple que se ejecuta 100 veces.

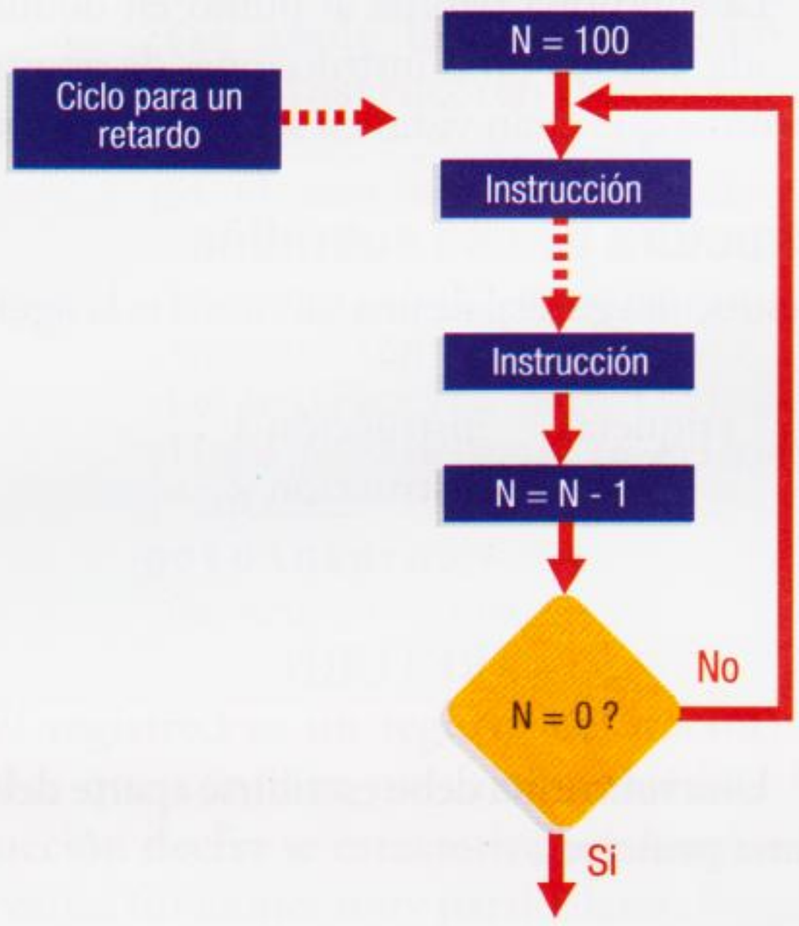


Figura 6.19 *Diagrama de flujo de una estructura repetitiva*

Las estructuras repetitivas se caracterizan por disponer de los siguiente elementos:

Condición inicial
Punto de retorno
Instrucción
. . .
Instrucción
Cambiar la condición inicial
Verificar la condición final
Si NO se cumple la condición final,
saltar al punto de retorno
Si se cumple la condición final,
continuar con la siguiente instrucción después del ciclo
Instrucción
. . .

Observe como se aplican estas ideas al diagrama de flujo de la **figura 6.19**.

Condición inicial (N = 100)
Punto de retorno
Instrucción (**Instrucción**)
. . .
Instrucción (**Instrucción**)
Cambiar condición inicial (N = N - 1)
Verificar la condición final
(¿N = 0?)
Si NO se cumple la condición final,
saltar al punto de retorno
(**Saltar a punto de retorno**)
Si se cumple la condición final,
continuar con la siguiente instrucción después del ciclo
Instrucción
. . .

Estas instrucciones en seudocódigo (instrucciones en español) representan otro punto de vista de lo que es una estructura repetitiva. Para algunas personas es bastante más claro un diagrama de flujo, aunque en realidad todo depende del punto de vista y del estilo del programador que esté desarrollando el trabajo.

Precaución

Es importante tomar en cuenta el siguiente punto:

> *Toda estructura repetitiva debe tener un mecanismo que permita cambiar la condición inicial. En caso de no hacerlo, el programa podría dar vueltas "para siempre", impidiendo que las acciones que vienen a continuación se ejecuten.*

El caso mencionado es posiblemente el punto más importante que se debe tener en cuenta a la hora de programar estructuras repetitivas. No respetar este principio puede llevar a errores que impidan el normal desarrollo del programa.

Subrutinas

Una subrutina es un conjunto de instrucciones que se ejecutan sólo si son llamadas desde otro programa. El concepto es el siguiente: un pro-

grama de microcontrolador ejecuta, en ciertos puntos de su desarrollo secuencial, un conjunto de instrucciones externas mediante el uso de una instrucción de llamado (CALL). A este paquete externo de instrucciones se lo conoce con el nombre de **subrutina.**

La **figura 6.20** presenta la estructura de una subrutina.

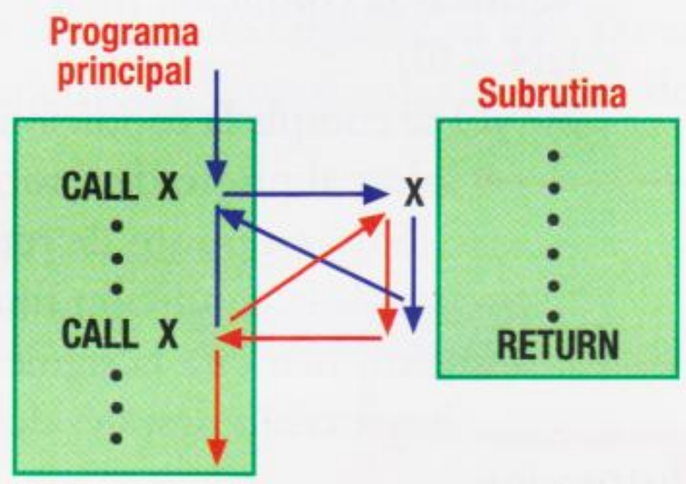


Figura 6.20 *Estructura de una subrutina*

¿Qué ventajas tienen las subrutinas? A continuación mencionamos algunas de las más importantes:

1. El programa es más fácil de leer y de escribir. La razón de esto se encuentra en que cada vez que se requieren procesos repetitivos se escribe una subrutina que contenga las instrucciones que deben repetirse, y se las llama cuando sea conveniente.
2. Al escribir una sola vez el conjunto de instrucciones se evita la posibilidad de cometer errores de digitalización.
3. Se ahorra un considerable espacio de memoria, ya que es más corto utilizar una instrucción de llamada, que escribir en su lugar todas las instrucciones contenidas en la subrutina.

Instrucción CALL

La instrucción que permite llamar a una subrutina es la instrucción CALL. Lo único que se requiere es indicar la dirección en donde se encuentra dicha subrutina. La sintaxis de la instrucción CALL es la siguiente:

CALL **etiqueta**

La etiqueta es una constante que se refiere a una posición en memoria y está ligada a la primera instrucción de la subrutina. Más adelante se verá como aplicar la instrucción CALL.

Instrucción RETURN

La instrucción RETURN permite retornar desde una subrutina. La sintaxis de un retorno de subrutina es la siguiente:

RETURN

La subrutina retorna al punto en donde fue llamada. Existen otras instrucciones de retorno de subrutina que serán vistas en su debido momento.

Estructura de una subrutina

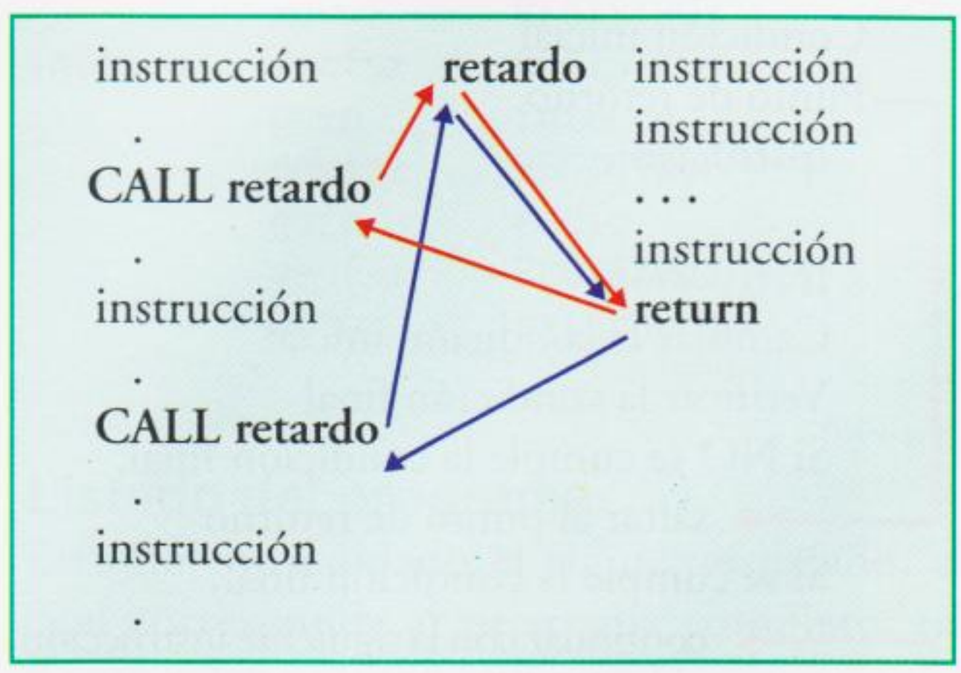
La estructura general de una subrutina es la siguiente:

```
etiqueta    instrucción 1
            instrucción 2
            .
            .
            RETURN
```

Una subrutina debe escribirse aparte del programa principal.

Llamados a una subrutina desde un programa

La apariencia que tendrá, en términos informales, el proceso de llamadas a subrutinas es la siguiente:



En este ejemplo se supone que la etiqueta **retardo** marca el punto en donde se encuentra la

primera instrucción de la subrutina. La última instrucción de la subrutina es **return**.

Instrucción de soporte

Para que los ciclos repetitivos se puedan ejecutar efectivamente, es necesario tener instrucciones que permitan verificar una determinada condición. El siguiente segmento de código ilustra este proceso.

```
interno  decfsz registro3
         ;Se resta uno al registro3.
         ;Si el valor resultante es cero
         ;(se cumple la condición), salta
         ;a la instrucción después de
         ;"goto".
         ;Si el contenido del registro3
         ;aún no es cero (no se cumple
         ;la condición), ejecuta la
         ;instrucción "goto".
         ;La instrucción goto (saltar)
         ;lleva nuevamente a la etiqueta
         ;"interno"
         goto interno
         ;Genera un ciclo
```

El **registro3** es un registro del microcontrolador que contiene un valor numérico. La instrucción **decfsz** se caracteriza por hacer, ella sola, varias funciones muy particulares. En primer lugar, resta el número uno al contenido del registro correspondiente, en este caso al **registro3**. Si el contenido del **registro3** aún no es cero, es decir, no se cumple la condición final, ejecuta la instrucción siguiente.

Dado que esta instrucción es un salto (goto) a la etiqueta **interno**, el programa entra a un ciclo repetitivo.

Si después de restar uno al **registro3** en alguna iteración del ciclo, se obtiene el valor cero, se habrá alcanzado la condición final. La instrucción **decfsz** genera un salto automático a la siguiente instrucción después de **goto interno**. Esto garantiza que cuando se alcance la condición final el ciclo terminará, y el programa podrá continuar su desarrollo normal.

Nota:
Todos los ciclos generan algún tipo de retardo.

La **figura 6.21** presenta el diagrama de flujo de la estructura repetitiva basada en el código presentado.

Figura 6.21 *Diagrama de flujo de las instrucciones que generan un retardo*

Registros de trabajo

Usualmente los ciclos (estructuras repetitivas) requieren el uso de registros para almacenar información que varía en el tiempo. Para nuestro proyecto vamos a definir los siguientes registros de trabajo:

```
REGISTRO1  EQU   0x0C
                 ;Primer registro de trabajo

REGISTRO2  EQU   0x0D
                 ;Segundo registro de trabajo

REGISTRO3  EQU   0x0E
           ;Tercer registro de trabajo
```

Control de estructuras repetitivas

El tiempo de un retardo generado por un ciclo repetitivo depende del valor inicial que se asigne al registro de trabajo. Para nuestro proyecto vamos a utilizar las siguientes constantes.

```
DATO1     EQU   25
                ;Dato usado para el
                ;retardo externo
DATO2     EQU   30
                ;Dato usado para el
                ;retardo medio
DATO3     EQU   35
                ;Dato usado para el
                ;retardo interno
```

Estructuras repetitivas anidadas

Como se verá a lo largo de su uso, las estructuras repetitivas por sí solas, a pesar de su potencia, en muchas ocasiones no son suficientes para lograr el objetivo propuesto. Por este motivo se acostumbra recurrir a ciclos dentro de ciclos, lo cual contribuye a aumentar el tiempo de retardo generado. Esta técnica de creación de retardos muchos la denominan **retardo por *software*** y ya la hemos utilizado en varias oportunidades en nuestra sección de proyectos. La idea de esta técnica es ejecutar muchas veces una serie de instrucciones que en realidad no hacen ninguna tarea importante, pero que sí emplean ciclos de máquina y por lo tanto tiempo.

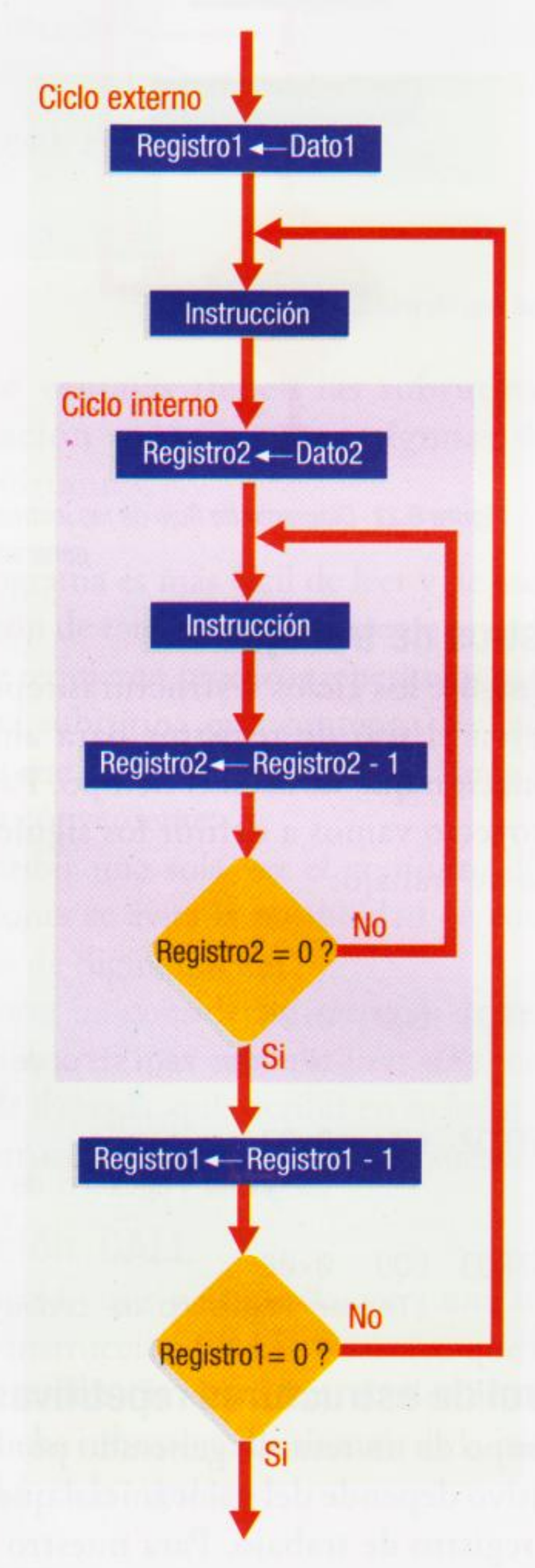


Figura 6.22 Estructuras repetitivas anidadas

Desde que la aplicación que nos encontremos desarrollando lo permita (cuando el microcontrolador entra a esta rutina, desatiende cualquier otra tarea a menos que ocurra una interrupción), podemos incorporar estas estructuras para lograr un retardo de cualquier tiempo. La **figura 6.22** muestra la estructura de un ciclo anidado.

Codificación de un ciclo anidado

A continuación se presenta la codificación en lenguaje ensamblador de un ciclo anidado.

```
retardo   movlw    DATO1
          movwf    REGISTRO1
externo   movlw    DATO2
          movwf    REGISTRO2
medio     movlw    DATO3
          movwf    REGISTRO3
interno   decfsz   REGISTRO3
          goto     interno
          decfsz   REGISTRO2
          goto     medio
          decfsz   REGISTRO1
          goto     externo
```

Listado del programa

Estudie con cuidado el siguiente listado, el cual corresponde al programa completo, tal y como debe introducirse en la memoria del microcontrolador, una vez convertido en código hexadecimal.

```
;=========================================
; PROGRAMA..: Pro05.asm
; AUTOR.....: CEKIT S.A.
; FECHA.....: Mayo de 2001
; COMENTARIO: El programa Pro05.asm enciende
             ;y apaga alternativamente un
             ;diodo LED.
;=========================================

;-----------------------------------------
; ENCABEZADO
;-----------------------------------------
      LIST       p=16F84     ;Se utiliza el
                             ;PIC 16F84
      RADIX      HEX         ;Se utiliza el
                             ;sistema hexadecimal
;-----------------------------------------
; ZONA DE ETIQUETAS
;-----------------------------------------
W       EQU     0       ; Si el destino
                        ;es W, debe utilizarse
                        ;el número 0
F       EQU     1       ;Si el destino es F,
                        ;debe utilizarse el
                        ;número 1
PUERTOB EQU     0x06    ;La posición 0x06
                        ;corresponde al puerto B
                        ;cuando el banco 0 está
                        ;seleccionado
TRISB   EQU     0x86    ;La posición 0x06
                        ;permite configurar el
                        ;puerto B como entrada
                        ;o como salida
ESTADO  EQU     0x03    ;El Registro de estado
REGISTRO1 EQU   0x0C    ;Registro de trabajo
REGISTRO2 EQU   0x0D    ;Registro de trabajo
REGISTRO3 EQU   0x0E    ;Registro de trabajo
DATO1   EQU     25      ;Dato usado para el
                        ;retardo externo
DATO2   EQU     30      ;Dato usado para el
                        ;retardo medio
DATO3   EQU     35      ;Dato usado para el
                        ;retardo interno
;-----------------------------------------
; INICIO DEL PROGRAMA
;-----------------------------------------
        ORG     0       ;Establecimiento
        GOTO    INICIO  ;del punto de inicio
        ORG     5       ;del programa
;-----------------------------------------
; CODIGO DEL PROGRAMA
;-----------------------------------------
inicio  bsf     ESTADO,5 ;Coloca un 1 en el bit
                         ;5 del registro de
                         ;estado. Esto selecciona
                         ;el banco 1 de conjunto
                         ;de registros de memoria
                         ;RAM
        clrf    TRISB    ;Cuando se almacenan ceros
                         ;en 0x06, estando
                         ;seleccionado el banco 1,
                         ;el puerto B se configura
                         ;como salida
        bcf     ESTADO,5 ;Coloca un 0 en el bit
                         ;5 del registro de estado.
                         ;Esto selecciona el
                         ;banco 0 del conjunto de
                         ;registros de memoria RAM
                         ;(ver mapa)
ciclo   movlw   1        ;Se coloca a 1 el bit de
                         ;menor peso del registro W.
                         ;En W se almacena
                         ;00000001
        movwf   PUERTOB  ;Se transfiere el contenido
                         ;del registro W al
                         ;puerto B. Esto significa
                         ;W -> PUERTOB.
                         ;Esta acción permite
                         ;energizar el diodo LED
                         ;conectado al bit 0 del
                         ;PUERTO B
        call    retardo  ;Antes de apagar el LED,
                         ;se llama a la subrutina
                         ;de retardo. Esto
                         ;garantiza que el diodo
                         ;LED permanecerá
                         ;encendido durante un
                         ;tiempo específico
        movlw   0        ;Se coloca a cero el
                         ;registro W.
                         ;En W se almacena 00000000
        movwf   PUERTOB  ;Se transfiere el contenido
                         ;del registro W al puerto
                         ;B. Esto significa W ->
                         ;PUERTOB.
                         ;Esta acción permite
                         ;apagar el diodo LED
                         ;conectado al bit 0 del
                         ;PUERTO B
        call    retardo  ;Antes de encender
                         ;nuevamente el diodo LED,
                         ;se llama a la subrutina
                         ;de retardo.
                         ;Esto garantiza que el
                         ;diodo LED permanecerá
                         ;apagado durante un tiempo
                         ;específico
        goto    ciclo    ;La instrucción "goto"
```

```
                                ;salta a la etiqueta "ciclo".
                                ;Esta instrucción de
                                ;salto permite repetir
                                ;nuevamente el proceso
                                ;de encendido y apagado
;-----------------------------------
; SUBRUTINA DE RETARDO
;-----------------------------------
; Las instrucciones que siguen sólo se
ejecutan si son llamadas con
; una instrucción "call"
;-----------------------------------

retardo movlw   dato1   ;Se carga W con el valor
                        ;en dato1 (25)
        movwf   registro1 ;Se traslada W al registro1
externo movlw   dato2   ;Se carga W con el valor
                        ;en dato2 (30)
        movwf   registro2 ;Se traslada W al registro2
medio   movlw   dato3   ;Se carga W con el valor
                        ;en dato3 (35)
        movwf   registro3 ;Se traslada W al registro3
interno decfsz  registro3 ;Se resta uno al registro3.
                        ;si el valor resultante
                        ;es cero salta a la
                        ;instrucción siguiente.
                        ;En caso contrario
                        ;continúa con la siguiente
                        ;instrucción. Esta
                        ;instrucción (goto) lo
                        ;hace regresar a la
                        ;etiqueta "interno"
        goto    interno ;Genera el ciclo "interno"
        decfsz  registro2 ;Si llega a esta
                        ;instrucción, es porque
                        ;el registro3 alcanzó el
                        ;valor cero. Ahora se
                        ;repite lo anterior
                        ;para el registro2, pero
                        ;esta vez saltando a
                        ;la etiqueta "medio"
        goto    medio   ;Genera el ciclo "medio"
        decfsz  registro1 ;Si llega a esta
                        ;instrucción, es porque
                        ;el registro2 alcanzó el
                        ;valor cero. Ahora se
                        ;repite lo anterior para
                        ;el registro1, pero esta
                        ;vez saltando a la
                        ;etiqueta "externo"
                        ;Sin embargo, ahora hay
                        ;una diferencia:
                        ;cuando el registro1
                        ;adquiere el valor cero,
                        ;ya no hay más instrucciones
                        ;de salto. El programa
                        ;continúa con la instrucción
                        ;"return" la cual hace
                        ;que el programa termine
                        ;el llamado a la subrutina
        goto    externo ;Genera el ciclo "externo"
;-----------------------------------
; RETORNO DE LA SUBRUTINA
;-----------------------------------
        return          ;Permite retornar de la
                        ;subrutina
;-----------------------------------
; FINAL DEL PROGRAMA
;-----------------------------------
        END             ;Fin del programa
```

Nota:

Para el registro TRISB se ha utilizado la dirección 0x86. Esto dirige al microcontrolador directamente al segundo banco de memoria.

Conclusión

La programación es un arte que se estructura por capas: cada vez que usted domina una técnica en particular, ésta le será útil para un problema de mayor complejidad.

Ejercicio No. 3 Lectura de una interrupción externa (INT)

Este tipo de interrupción trabaja detectando los flancos de una señal de entrada por un pin determinado, el cual se puede ubicar fácilmente por poseer la denominación **INT** en el diagrama de las hojas de datos. Como se mostró en la sección de teoría, el tipo de flanco debe seleccionarse mediante el *bit* 6 del registro OPCION (INTEDG), siendo las posibilidades la detección de flancos de subida o flancos de bajada.

El procedimiento exacto que sigue un microcontrolador, desde el momento en que aparece la interrupción, es el siguiente. Si se encuentra ejecutando una instrucción la CPU la termina de ejecutar completamente, lee la bandera indicadora del habilitador global (GIE) y si la en-

cuentra activada quiere decir que las interrupciones se encuentran habilitadas. Ahora debe detectar si la bandera indicadora de interrupción INT se encuentra activada, si es así, procede a la secuencia de interrupción. Si alguna de las dos banderas anteriores se encuentra desactivada, el microcontrolador hará caso omiso de las interrupciones que ocurran y continuará con su trabajo normalmente.

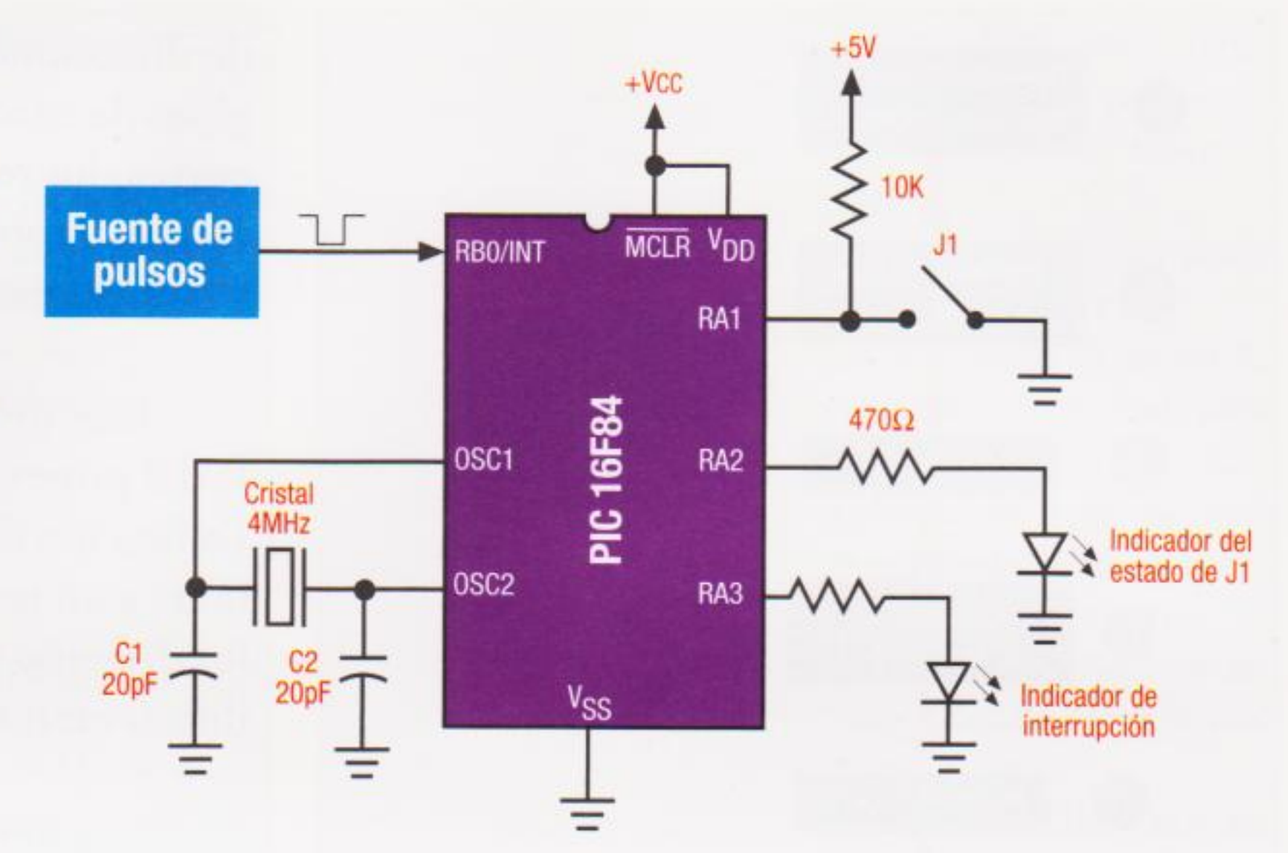


Figura 6.23 *Circuito básico para trabajar con una interrupción externa INT*

Asumiendo que se detectaron correctamente ambas banderas el microcontrolador borra la bandera del habilitador global, esto para evitar reconocer interrupciones adicionales, luego guarda la dirección de retorno en la pila y salta a la dirección 04.

El circuito de la **figura 6.23** nos servirá como ejercicio para detectar una interrupción de este tipo y familiarizarnos con su manejo, y las posibilidades que podemos tener.

En la **figura 6.24** observamos el código fuente del ejercicio y en la **figura 6.25** su diagrama de flujo, con estos dos elementos podemos tener una idea muy completa del desarrollo del ejercicio.

En este ejercicio el microcontrolador lee el *jumper* J1 que se encuentra conectado al pin 1 del puerto A y muestra su estado en el pin 2 del puerto A; en esta operación permanece hasta que aparece una interrupción externa.

Cuando aparece una interrupción externa el microcontrolador salta a la posición 04, donde se encuentra la subrutina de la interrupción. Ella activa el LED indicador ubicado en el pin 3 del puerto A como señalización de que ha ocurrido dicha interrupción.

Descripción del programa

En el código fuente mostrado en la figura 6.24, podemos observar las partes fundamentales que

```
; EJERCICIO: LECTURA DE UNA INTERRUPCIÓN EXTERNA
;------ A -----------
      list   p=16f84
      radix   hex
;------ B -----------
W            EQU    0
F            EQU    1
PUERTOA      EQU    05
INTCON       EQU    0B
;------ C -----------
      ORG    0
      GOTO   INICIO
      ORG    4
      GOTO   INTERRUPCION
;------ D -----------
INICIO
      MOVLW  B'01111111'
      OPTION
      BCF    INTCON,1      ;BORRAR BANDERA INT
      BSF    INTCON,7      ;ACTIVAR HABILITADOR
GLOBAL
      BSF    INTCON,4      ;HABILITAR INTERRUPCIÓN
INT
      MOVLW  B'00000010'
      TRIS   PUERTOA       ;CONFIGURAR PUERTO A
      BCF    PUERTOA,2     ;INICIALIZAR PINES 2 Y 3
      BCF    PUERTOA,3     ;DEL PUERTO A
;------ E -----------
PRUEBA BTFSS PUERTOA,1     ;LEO EL JUMPER J1
      GOTO   ETIQUETA1     ;
ETIQUETA2                  ;SI ESTÁ DESACTIVADO
      BSF    PUERTOA,2     ;SE PONE EN ALTO EL PIN 2
      GOTO   PRUEBA        ;DEL PUERTO A
ETIQUETA1                  ;SI ESTÁ ACTIVADO
      BCF    PUERTOA,2     ;SE PONE EN BAJO EL PIN 2
      GOTO   PRUEBA        ;DEL PUERTO A
;------ F -----------
INTERRUPCION
      BSF    PUERTOA,3     ;SE PONE EN ALTO EL PIN 3
CICLO GOTO   CICLO         ;DEL PUERTO A
      END
```

Figura 6.24 *Código fuente del ejercicio de lectura de una interrupción externa*

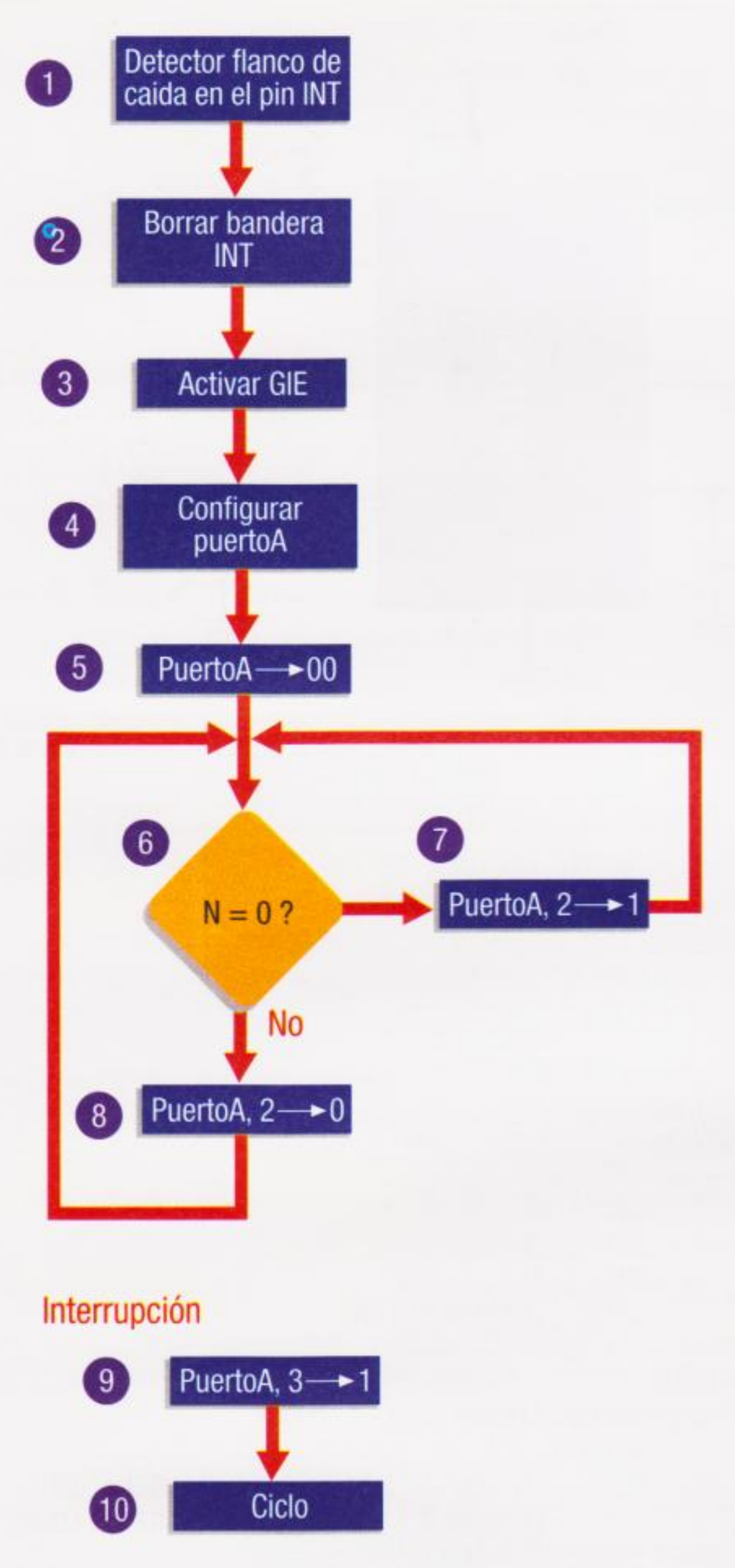


Figura 6.25 *Diagrama de flujo del ejercicio de lectura de una interrupción externa*

componen un programa en lenguaje ensamblador. En la parte etiquetada con la letra A se han escrito dos directivas que simplemente sirven para establecer algunas configuraciones que el programa ensamblador utilizará para crear el código objeto; en este caso se declara que se va a trabajar con un PIC16F84 y que el formato por defecto de los valores constantes es el hexadecimal.

Siempre que desarrollamos un programa para microcontrolador, debemos especificar el mapa de memoria RAM que se va a manejar. Este mapa de direcciones debe contener los puertos, el registro de estado, el registro de interrupciones, etc.; esto es, los registros especiales o SFR y los registros de propósito general que vamos a utilizar para nuestras variables.

Esas definiciones están dadas en la sección B del programa a través de las directivas EQU. La función de esta directiva es definir una constante a un nombre en el programa; cualquier valor definido por EQU es fijo y no se puede modificar en ninguna parte del programa.

Específicamente para este ejercicio, solo necesitamos definir las direcciones del puerto A y de INTCON que son los registros que vamos a utilizar; al registro de trabajo W se ha asignado el número 1 y a la letra F el número 0, de esta manera, dentro del programa, cada que pongamos la letra W, el microcontrolador entenderá que equivale a poner un 1.

En el bloque C del programa establecemos las direcciones dentro de la memoria de programa, del inicio del programa y del código de la rutina de interrupción, a través de la directiva ORG. Recordemos que en la lección de organización de la memoria observamos que efectivamente la rutina de interrupción se ubica en la dirección 04.

En el bloque D es donde realmente se inicia nuestro programa. En primer lugar configuramos adecuadamente el registro OPCION; para llevar a cabo esta tarea debemos cargar W con el número deseado y teclear la instrucción OPTION.

En este caso W se ha cargado mediante la instrucción **MOVLW B'01111111'**. Ella carga en W el número B'0111111' en donde la B antes del número quiere decir que se encuentra en notación binaria. Si nos basamos en el mapa del registro OPCION que vimos en la lección de la ALU y la CPU, podemos decir que se van a leer flancos de bajada para la interrupción externa debido al 1 en el *bit* 6 (INTEDG).

Ahora debemos trabajar con el registro INTCON que vimos en la lección de teoría de manejo de interrupciones. Para que el microcontrolador esté en capacidad de manejar la interrupción externa debemos habilitarla, poniendo un 1 en los *bits* 4 y 7 de INTCON, que corresponden al habilitador de la interrupción INT y al habilitador global de las interrupciones (GIE). Finalmente, dentro del mismo bloque D configuramos el funcionamiento del puerto A para que opere de acuerdo al circuito planteado en este ejercicio. A partir de este momento, el microcontrolador estará a la espera de un flanco de bajada por el pin INT para generar una interrupción y saltar a la subrutina correspondiente. Los numerales 1 al 5 del diagrama de flujo corresponden a este bloque.

En el bloque E se encuentra el programa principal del ejercicio. Éste se ejecutará normal y continuamente hasta que no se presente el flanco de bajada en el pin INT. En la línea donde se encuentra la etiqueta **PRUEBA** se hace una lectura del pin 1 del puerto A que es donde se encuentra el *jumper* de entrada. Para ello utilizamos la instrucción **BTFSS PUERTOA,1**, que pregunta si el *bit* 1 del puerto A se encuentra en 1 y se salta la siguiente instrucción si la respuesta es afirmativa. De acuerdo al estado de dicho *bit*, saltamos a **ETIQUETA1** ó a **ETIQUETA2** donde se activa o desactiva el LED, de acuerdo a la lectura del *jumper*. En el diagrama de flujo este procedimiento lo podemos apreciar desde el bloque 6 hasta el bloque 8, donde podemos ver cada uno de los pasos seguidos.

En el bloque F, numeral 9, se encuentra la subrutina de interrupción. Cuando el microcontrolador detecta el flanco correspondiente por el pin INT, se dirige hasta este bloque del programa. En este punto el microcontrolador simplemente activa su salida por el pin 3 del puerto A, señal con la que nos damos por enterados que se ha entrado a la rutina de interrupción.

Copie este programa en un editor de texto, ensámblelo a través del programa MPASMWIN (disponible en *www.microchip.com*) y prográmelo en un microcontrolador PIC16F84, utilizando nuestro programador K-224 o cualquier otro. Monte el circuito en un *protoboard* y haga todas las pruebas y variaciones que se le ocurran. Recuerde que es la práctica la que hace al maestro.

Instrucción GOTO

El contador de programa salta a la dirección "k". No afecta a ningún *bit* del registro de estado.

Sintaxis: GOTO k

Ejemplo:

```
        GOTO      ciclo
```

Antes de la instrucción
Contador de programa = ?
Después de la instrucción
Contador de programa = ciclo

Instrucción BTFSS

Sintaxis: BTFSS f,b

Prueba el *bit* "b" del registro "f" y salta una línea si está en uno.

Ejemplo:

```
aquí   BTFSS    regis,0
falso  GOTO     .......
verdad......
```

Antes de la instrucción
Contador de programa = aquí
Después de la instrucción
Si el *bit* 0 del registro regis = 1
Contador de programa = verdad
Si el *bit* 0 del registro regis = 0
Contador de programa = falso

Instrucción BTFSC

Sintaxis: BTFSC f,b

Prueba el *bit* "b" del registro "f" y salta una línea si está en cero.

Ejemplo:

```
aquí   BTFSC    regis,0
falso  GOTO     ....
verdad......
```

Antes de la instrucción
Contador de programa = aquí
Después de la instrucción
Si el *bit* 0 del registro regis = 0
Contador de programa = verdad
Si el *bit* 0 del registro regis = 1
Contador de programa = falso

Ejercicio No. 4. Generación de señales digitales a través del TMR0

En este ejercicio vamos a utilizar las propiedades del temporizador interno de los PIC denominado TMR0 para generar una onda cuadrada que se entrega por un pin del microcontrolador y se aplica sobre un diodo LED, **figura 6.26**. Como fuente de señal para el temporizador se utilizan los ciclos de reloj internos del microcontrolador, sometidos a un proceso de división con el módulo prescalador con un factor de 256. Es decir, vamos a dividir la frecuencia interna por 256 y con esa frecuencia resultante vamos a incrementar secuencialmente el registro del TMR0.

En la **figura 6.27** podemos apreciar el diagrama de flujo del ejercicio; inicialmente se configura el funcionamiento del TMR0 por medio del registro OPCION, se configura el puerto A como salida y luego se entra a un ciclo que basa su funcionamiento en el resultado del TMR0. En primera instancia se activa el LED de salida permaneciendo en ese estado durante un tiempo determinado, luego se desactiva el LED permaneciendo apagado durante el mismo periodo de tiempo que estuvo encendido; este ciclo se repite de manera indefinida.

La parte más interesante del programa es la subrutina de retardo desarrollada alrededor del TMR0, la cual vemos en la **figura 6.28**.

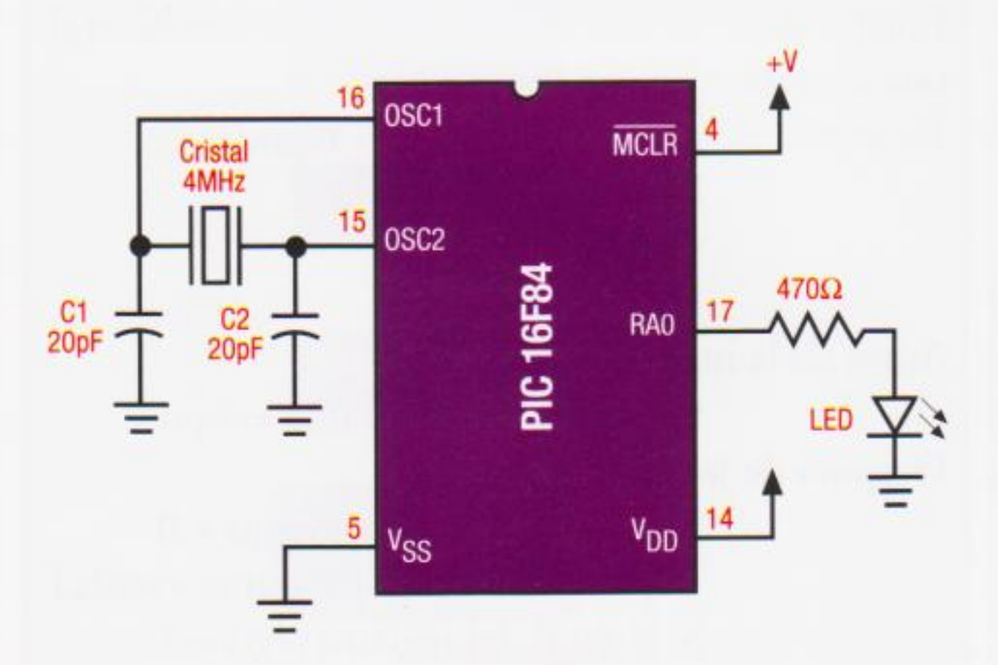


Figura 6.26 *Diagrama esquemático del ejercicio*

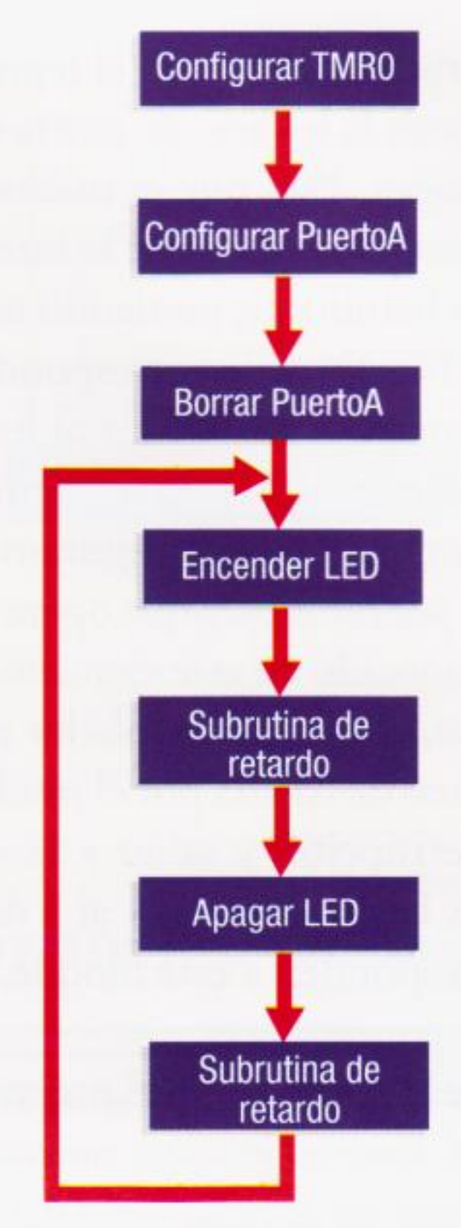


Figura 6.27 *Generación de señales digitales mediante el TMR0*

Figura 6.28 *Subrutina de retardo*

Cada vez que se invoca la subrutina RETARDO, la primera acción que se ejecuta es un borrado del registro TMR0, lo que hace que el temporizador empiece su incremento desde cero. Cuando el TMR0 trabaja en modo temporizador, quiere decir que se incrementa automáticamente con los pulsos internos del reloj divididos por el factor establecido con el prescalador. En esta subrutina se hace un chequeo constante del *bit* 6 del registro TMR0, que se man-

tiene hasta que dicho *bit* se ponga en uno; el tiempo transcurrido para que el *bit 6* se ponga en uno es el que vamos a utilizar para generar el retardo.

Para calcular el retardo procedemos de la siguiente forma: si utilizamos un cristal de 4 Mhz la frecuencia interna es de 1Mhz, ya que se divide por cuatro (debido a los ciclos de búsqueda y ejecución que estudiamos en la sección de teoría, lecciones sobre la CPU). Esta frecuencia interna se debe dividir por 256 que es el valor configurado con el prescalador lo que nos da un periodo de 256 us. Este valor multiplicado por 65, que es el valor hasta el cual cuenta el TMR0 para llegar al *bit* 6, nos da un periodo final de 16.64 ms.

En la práctica, el LED parece estar encendido todo el tiempo debido a que 16.64 ms es un tiempo muy corto para ser detectado por el ojo humano. La verdadera forma de esta onda puede apreciarse en un osciloscopio.

En la **figura 6.29** podemos apreciar el valor que se debe asignar al registro OPCION para cumplir los requerimientos del ejercicio; si tiene alguna duda sobre el significado de cada uno de estos *bits*, consulte la sección de teoría donde se hace una descripción completa del registro OPCION. Finalmente en la **figura 6.30** podemos apreciar el código fuente del ejercicio.

Ejercicio No. 5. Contando eventos externos

En el anterior ejercicio utilizamos el TMR0 en el modo de temporizador; esto significa que se incrementaba con los ciclos internos de reloj. En este ejercicio vamos a trabajar con la segunda opciónpara el manejo del TMR0 la cual corresponde al modo de contador. Ahora, el registro del TMR0 se incrementará sólo con la aparición de eventos externos, por un pin especialmente asignado para esta labor (pin TOCKI, RA4 en el 16F84). El diagrama esquemático del circuito lo podemos apreciar en la figura **6.31**. El objeto en este ejercicio es simplemente leer la entrada de pulsos externos, incrementar el TMR0 de acuerdo a los pulsos que se han leído y visualizar el valor almacenado en el temporizador por medio de ocho diodos LED conectados en el puerto B.

Figura 6.29 *Configuración del registro OPCION*

```
;=========================================
; EJERCICIO DE TEMPORIZACION CON EL TMR0
; SE CHEQUEA EL BIT 6 DEL TMR0 RESULTANDO 65
; INCREMENTOS
; RETARDO = 16.64 MILISEGUNDOS
; CRISTAL 4 MHZ
;-------------------
      LIST  P=16F84
      RADIX HEX
;-------------------
W     EQU   0
F     EQU   1
;-------------------
TMR0  EQU   01
PORTA EQU   05
PORTB equ   06
CONTA EQU   0D
;-------------------
      ORG   0

INICIO
      CLRWDT            ;CONFIGURAR OPCION
      MOVLW B'11010111' ;ASIGNAR PRESCALADOR
      OPTION            ;EN 256, E INCREMENTO POR
                        ;RELOJ INTERNO
      MOVLW 0X00
      TRIS  PORTA       ;PUERTO A COMO SALIDAS
      CLRF  PORTA       ;BORRO EL PUERTO

ONDA  BSF   PORTA,0     ;ENCIENDO EL LED
      CALL  RETARDO     ;RETARDO POR SUBRUTINA
      BCF   PORTB,0     ;APAGO EL LED
      CALL  RETARDO     ;RETARDO POR
                        ;SUBRUTINA
      GOTO  ONDA

RETARDO     CLRF  TMR0  ;INICIO LA
                        ;CUENTA DESDE CERO

REPITO      BTFSS TMR0,6 ;¿BIT 6 DEL TMR0 EN UNO?
      GOTO  REPITO      ; NO -> BORRO DE NUEVO
      RETURN            ; SI -> SALGO DE LA
                        ; SUBRUTINA

      END
```

Figura 6.30 *Código para la generación de señales digitales*

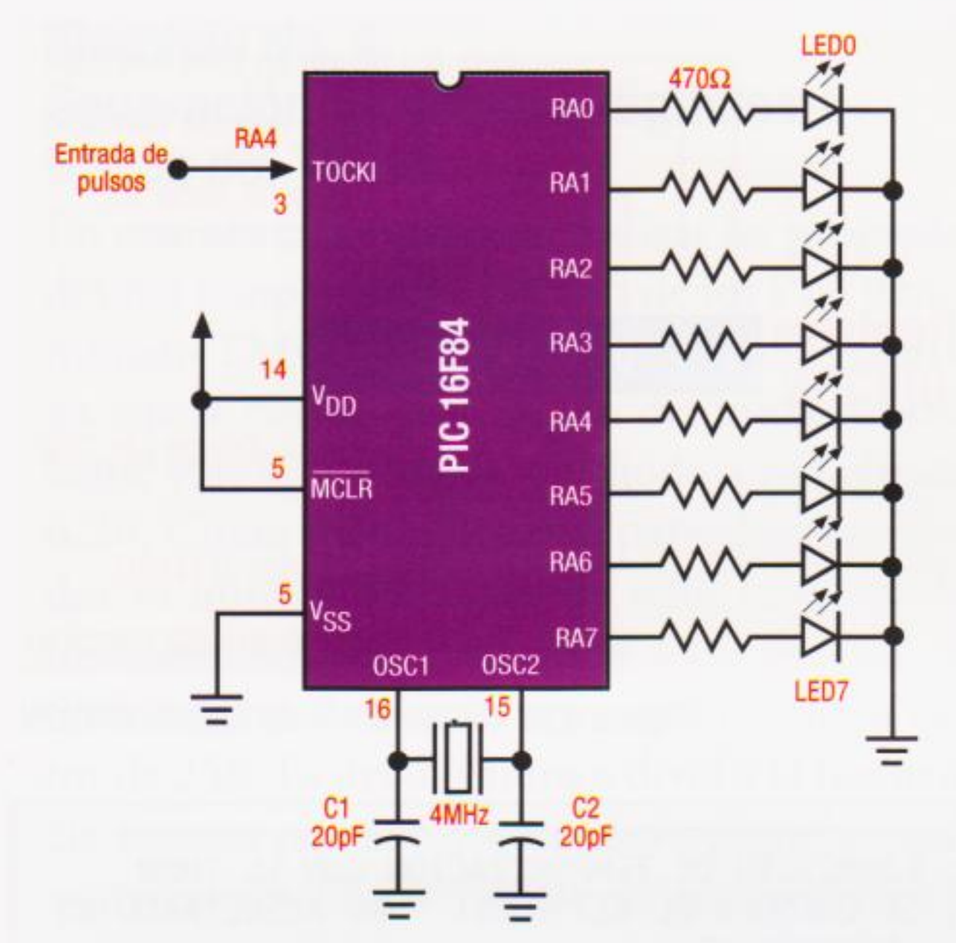


Figura 6.31 *Diagrama esquemático del ejercicio*

Figura 6.33 *Configuración del registro OPCION para contar eventos externos*

En la **figura 6.32** podemos apreciar el diagrama de flujo del presente ejercicio, en la **figura 6.33** la configuración del registro OPCION para este caso y en la **figura 6.34** el código fuente. La principal diferencia con respecto al ejercicio anterior, es la configuración establecida en el registro OPCION, la cual brinda la directriz sobre cual es la fuente de señal que va a utilizar el TMR0; en este caso son los pulsos externos y sobre ellos no ejecuta su acción el prescalador debido a que ese módulo se ha evitado al asignarlo al temporizador *watchdog*.

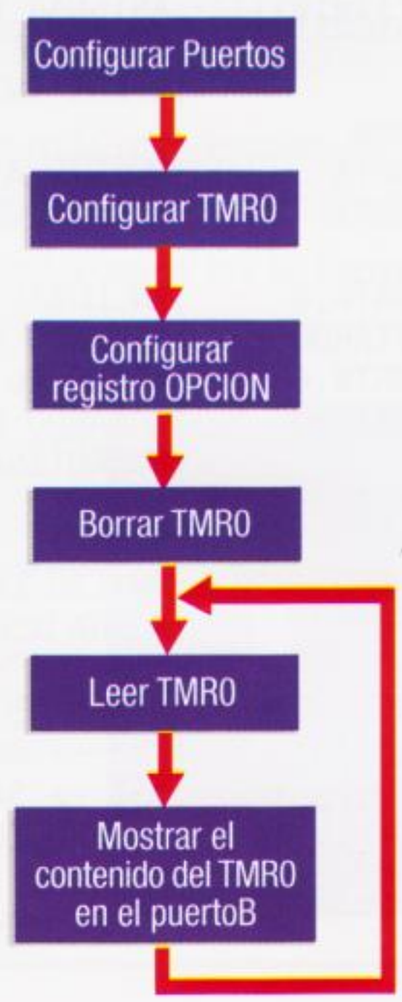


Figura 6.32 *Diagrama para contar eventos externos*

```
;-------------------------
; EJERCICIO PARA CONTAR EVENTOS EXTERNOS
;-------------------------

      LIST  P=16F84
      RADIX HEX
;-------------------------
W     EQU   0
F     EQU   1
TMR0  EQU   01
PORTA EQU   05
PORTB EQU   06
;-------------------------
      ORG   0

INICIO
      MOVLW 0XFF   ; CONFIGURAR PUERTO A
      TRIS  PORTA  ; COMO ENTRADAS
      MOVLW 0X00   ; CONFIGURAR PUERTO B
      TRIS  PORTB  ; COMO SALIDAS
      CLRF  PORTB  ; INCIAR EL PUERTO B
      CLRF  TMR0   ; INCIAR EL TMR0
      CLRWDT       ; DESACTIVAR WATCHDOG
      MOVLW B'01101111'; CONFIGURAR EL
      OPTION             ; REGISTRO TMR0
      CLRF  TMR0   ; INICIAR DESDE CERO DE NUEVO

CICLO
      MOVF  TMR0,W ; LEER EL TMR0
      MOVWF PORTB  ; VISUALIZAR RESULTADO EN EL
                   ; PUERTO B
      GOTO  CICLO
      END
```

Figura 6.34 *Código fuente del ejercicio No.5.*

CLRWDT

Borra la cuenta del temporizador *watchdog*

Sintaxis: CLRWDT

Ejemplo: CLRWDT

Antes de la instrucción

Contador WDT = ?

Después de la instrucción

Contador WDT = 00

RETURN

Retorno desde subrutina

Sintaxis: RETURN

Ejemplo: RETURN

Después de la subrutina, contador de programa = pila

Ejercicio No. 6
Generación de señales mediante el TMR0 e interrupciones

En este ejercicio vamos a estudiar otra técnica para generar señales digitales; en el ejercicio No. 4 observamos que cada vez que el registro TMR0 llegaba hasta cierto valor se generaba cada uno de los tiempos. En esta ocasión vamos a cargar un valor inicial en el TMR0 a partir del cual se incrementa el registro mediante los ciclos de reloj internos y al desbordarse, genera una interrupción.

El diagrama esquemático de este ejercicio lo podemos apreciar en la **figura 6.35** y su montaje práctico sobre un circuito impreso universal en la **figura 6.36**.

Estando dentro del servicio de interrupción, se chequea el estado del pin de salida. Si está en estado lógico alto, se pone en bajo y viceversa.

El cuerpo principal de este programa lo podemos apreciar en el diagrama de flujo de la **figura 6.37**. Inicialmente se hacen las configuraciones básicas para el manejo que se va a hacer sobre las interrupciones y el modo de trabajo del TMR0; se carga un valor específico en el registro TMR0 y se entra en un ciclo de espera para el momento en que se presente la interrupción.

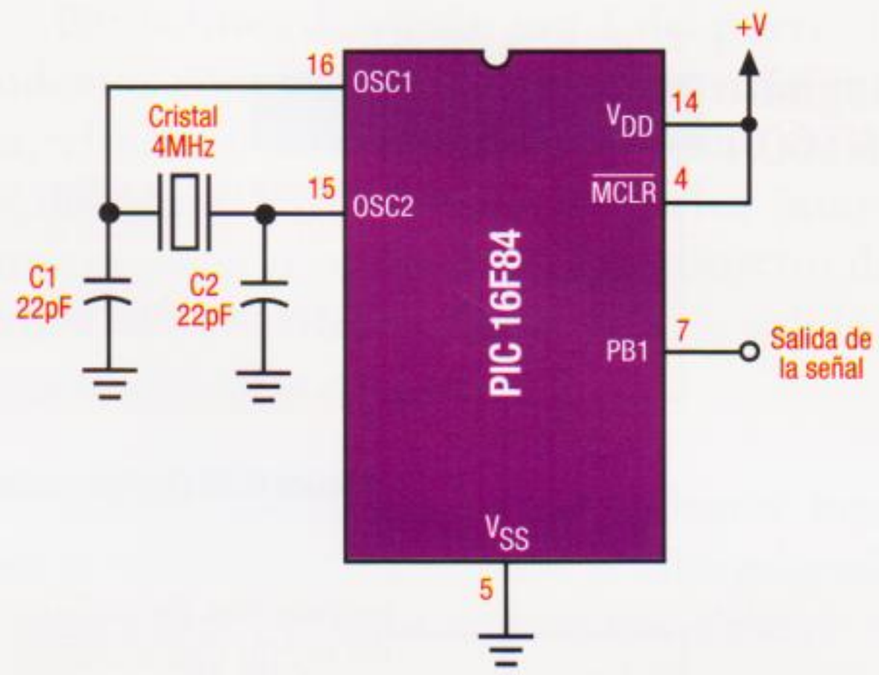


Figura 6.35 *Diagrama esquemático del ejercicio*

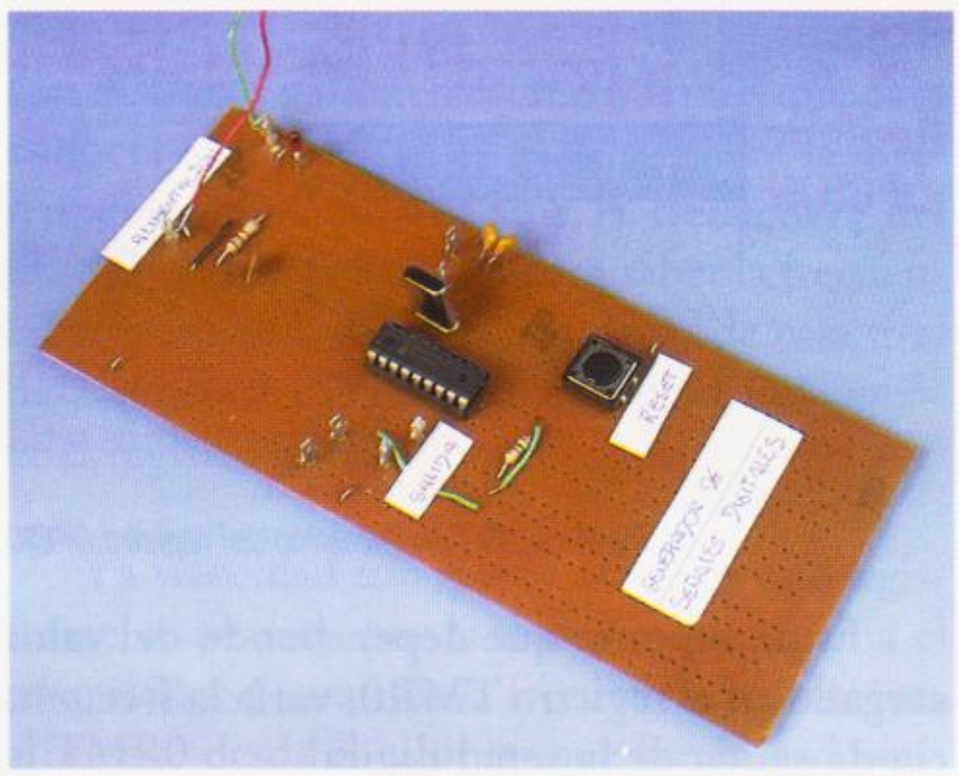


Figura 6.36 *Montaje sobre un circuito impreso universal*

Figura 6.37 *Diagrama de flujo del ejercicio*

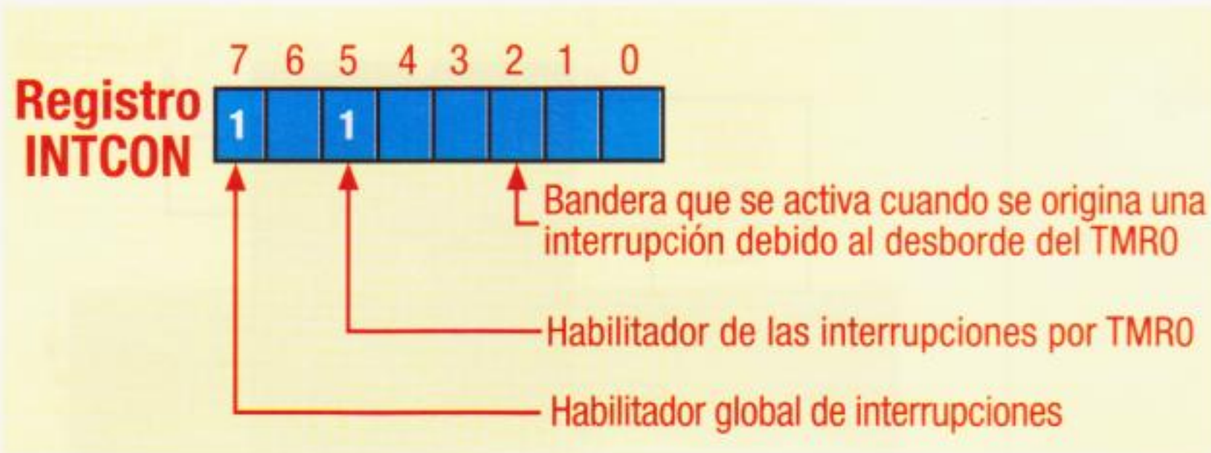


Figura 6.38 *Configuración del registro INTCON*

Figura 6.39 *Configuración del registro OPCION*

Es de suponer que dependiendo del valor cargado en el registro TMR0, varía la frecuencia de salida de la señal digital.

Como hemos visto, la configuración de las interrupciones se hace por medio del registro INTCON (**INT**errupt **CON**trol), **figura 6.38**; el primer paso es borrar la bandera TOIF (*bit* 2), la cual se activa cada vez que se presenta una interrupción originada por un desborde del registro TMR0.

Esta precaución también se debe tener en cuenta dentro de la rutina de interrupción; mientras esta bandera permanezca activa no se podrán atender nuevas interrupciones del TMR0.

Finalmente se deben activar el *bit* habilitador global de interrupciones (GIE) y el *bit* habilitador de la interrupción por desborde del TMR0. Con el cumplimiento de estas dos condiciones, se activa en forma definitiva la generación de la interrupción y ya solo basta esperar el momento en que ella ocurra para ejecutar la subrutina denominada **servicio de interrupción**.

La configuración del TMR0 se hace mediante el registro OPCION; en él se establece que se va a trabajar con los ciclos internos de reloj y el valor que se va a tomar para el prescalador, **figura 6.39**.

A partir de ese momento el programa espera que se presente el desborde del TMR0 y por lo tanto la interrupción. Cuando ella llega se ejecuta la estructura mostrada en la **figura 6.40**, en donde se pregunta por el estado actual de la línea de salida y se origina una salida que invierte dicha lectura. El código fuente de este ejercicio es el que se muestra en la **figura 6.41**.

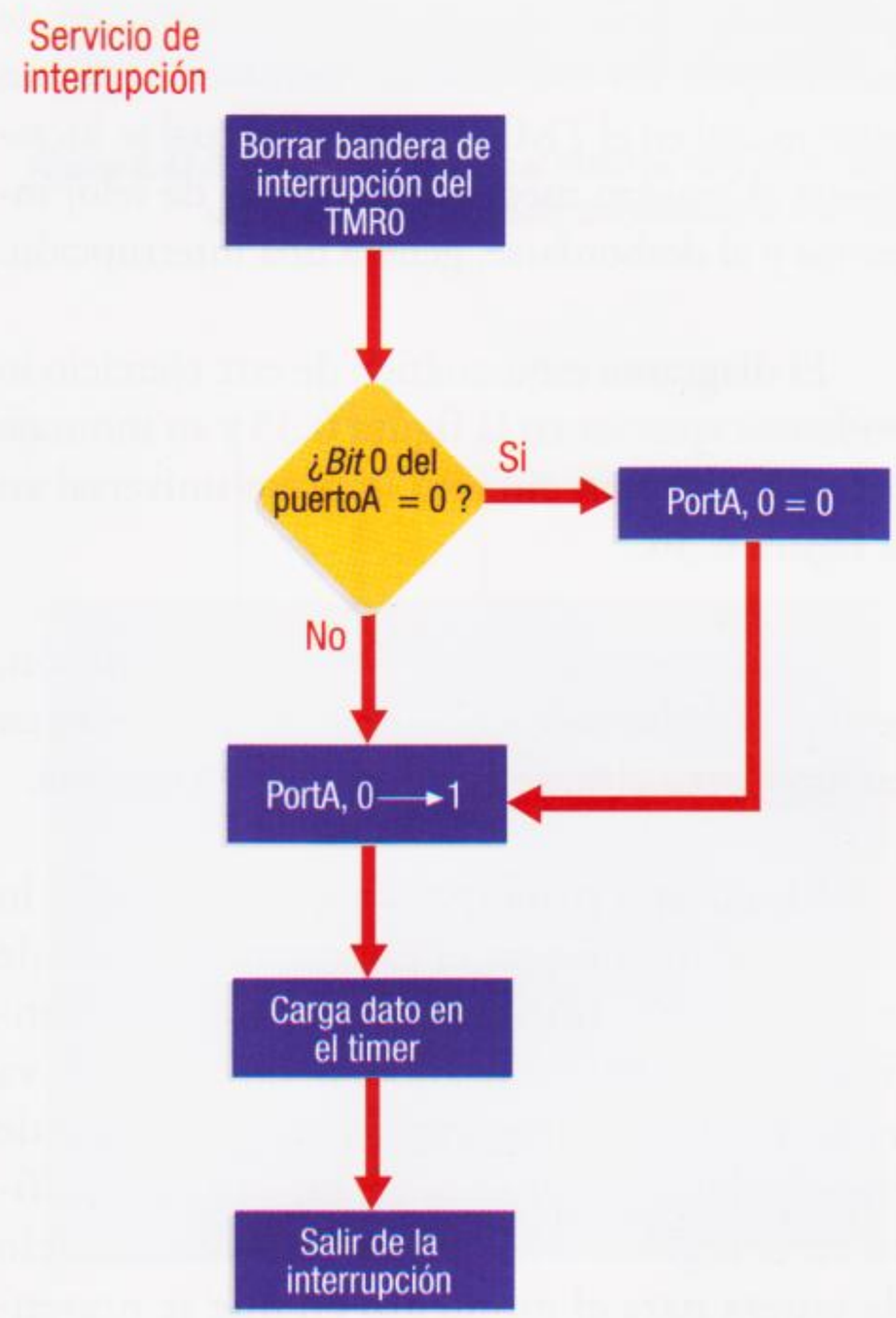


Figura 6.40 *Diagrama de flujo del servicio de interrupción*

```
;=====================EJERCICIO 6==================
;==================================================
; DEMOSTRACIÓN DEL USO DEL TIMER0
;
        LIST   P=16F84
        RADIX  HEX
;-------------------------------
W       EQU    0
F       EQU    1
;-------------------------------
; MAPA DE MEMORIA
TMR0    EQU    01
PORTB   EQU    06
INTCON  EQU    0B
;-------------------------------
        ORG    0
        GOTO   INICIO
        ORG    4
        GOTO   SER_INT

INICIO  MOVLW  B'00000000'   ; CONFIGURAR PINES DEL
        TRIS   PORTB         ; PUERTO B COMO SALIDAS
        CLRF   PORTB
        BCF    INTCON,2      ; BORRAR BANDERA DE INT.
                             ; DEL TMR0
        BSF    INTCON,7      ; HABILITADOR GLOBAL DE
                             ; INTERRUPCIONES
        BSF    INTCON,5      ; HABILITAR INTERRUP-
                             ; CIONES DEL TMR0
        CLRWDT
        MOVLW  B'01010110'   ; CONFIGURAR TMR0
        OPTION
        MOVLW  0X00
        MOVWF  TMR0
CICLO   GOTO   CICLO

;-----------SERVICIO DE INTERRUPCION----------------
SER_INT
        BCF    INTCON,2      ; BORRAR BANDERA DE INT. DEL
                             ; TMR0. ESTO PERMITE LEER
                             ; FUTURAS INTERR.
        BTFSS  PORTB,1       ; ¿PIN 1 DEL PUERTO B EN UNO?
        GOTO   ACTIV_BIT     ; SI ESTA EN CERO SE DEBE
                             ; ACTIVAR
BORRA_BIT
        BCF    PORTB,1       ; PONER EN BAJO EL BIT 1
                             ; DEL PUERTO B
        GOTO   CONFIG_TMR0
ACTIV_BIT
        BSF    PORTB,1       ; PONER EN ALTO EL BIT 1
                             ; DEL PUERTO B.
CONFIG_TMR0
        MOVLW  0X00          ; CARGAR ESTE VALOR PARA
                             ; INICIAR DESDE ALLÍ
        MOVWF  TMR0          ; INICIAR EL TMR0
        RETFIE
        END
```

Figura 6.41 *Código fuente del ejercicio No. 6*

Por la línea de salida, pin 1 del puerto B, podemos conectar un diodo LED o aún mejor, visualizar el resultado por medio de un osciloscopio. Este ejercicio se vuelve interesante cuando jugamos con los parámetros del sistema para variar la frecuencia de salida y llegar a un valor deseado.

Los parámetros con los que podemos jugar son: el valor del cristal externo, el valor asignado al prescalador y el valor cargado inicialmente en el registro TMR0.

En las **figuras 6.42, 6.43, 6.44** y **6.45** podemos apreciar diversas frecuencias generadas con distintos parámetros; recordemos que para conocer el momento en que se genera la interrupción, debemos dividir la frecuencia del cristal externo por cuatro para saber la frecuencia interna del microcontrolador; de esta manera, sabemos cada cuanto tiempo se incrementa el registro TMR0.

La velocidad con que se aumenta este registro se ve alterada por la división que origina el prescalador y esta nueva frecuencia incrementa al TMR0 desde el valor en que fue cargado inicialmente. Cuando llega a FF y pasa a 00 se origina una condición de desborde y se genera la interrupción donde se modifica la salida.

Con este ejercicio culminamos esta serie de ejercicios básicos en microcontroladores PIC, al igual que el aprendizaje sobre el módulo temporizador TMR0, que es uno de los más importantes dentro de los microcontroladores.

Hasta ahora hemos visto como utilizar un microcontrolador PIC como contador y como temporizador, al igual que se ha mostrado como modificar todos los parámetros de su funcionamiento. Ya estamos listos para abordar una nueva lección de ejercicios de un nivel un poco más elevado, donde desarrollaremos ejercicios de comunicaciones seriales, manejo de módulos LCD, conexión de memorias seriales, etc.

Frecuencia = 250 Hz

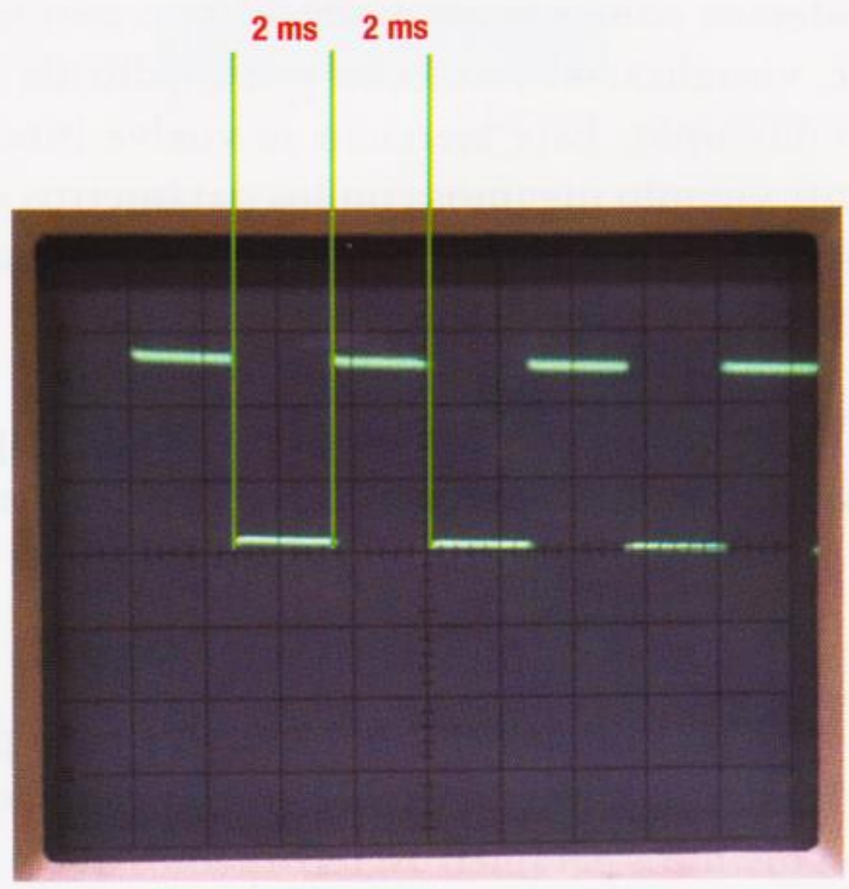


Figura 6.42 *En este caso se ha cargado el TMR0 con el valor hexadecimal F1 y se ha asignado un factor de división de 128*

Frecuencia = 1 KHz

Podemos notar que entre mayor sea el factor de división del TMR0, se necesitará un tiempo mayor para llegar al desborde, por lo tanto un período mayor.

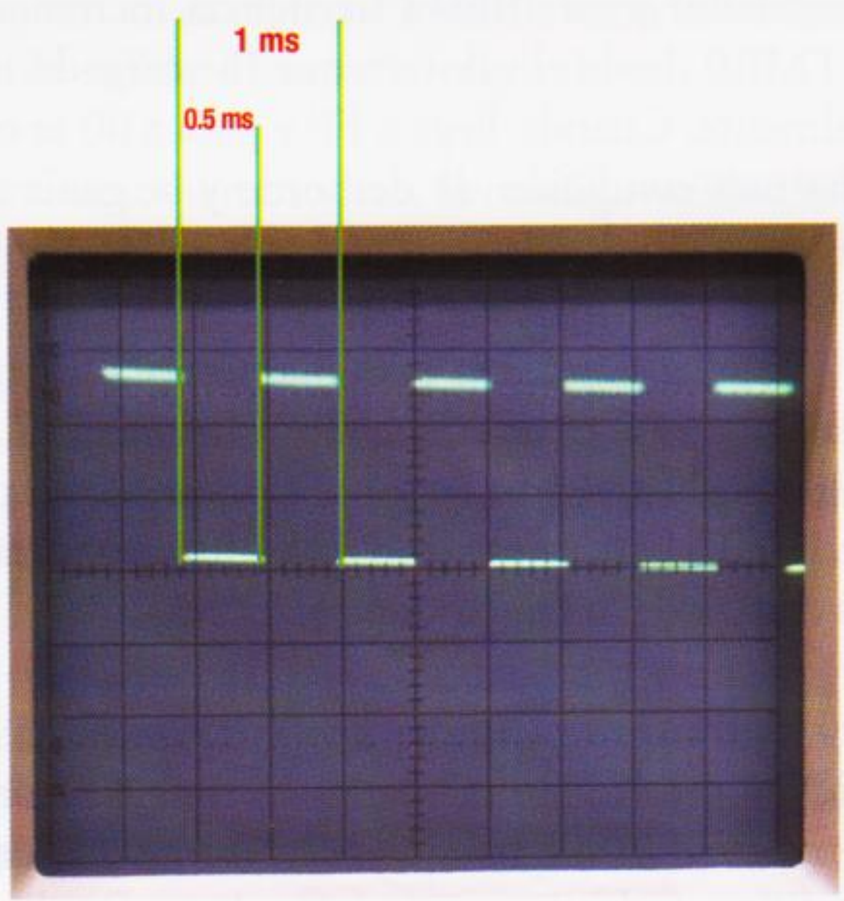


Figura 6.44 *En este caso también se ha borrado el TMR0 pero se ha hecho una división por 2, cargando el prescalador con 000 en los tres primeros bits del registro OPCION*

Frecuencia = 1,6 KHz

Cada vez que el TMR0 se desborda se genera una interrupción en la cual se cambia el estado de la señal de salida. El instante en que ocurre un cambio de nivel es el momento exacto en que la interrupción es atendida.

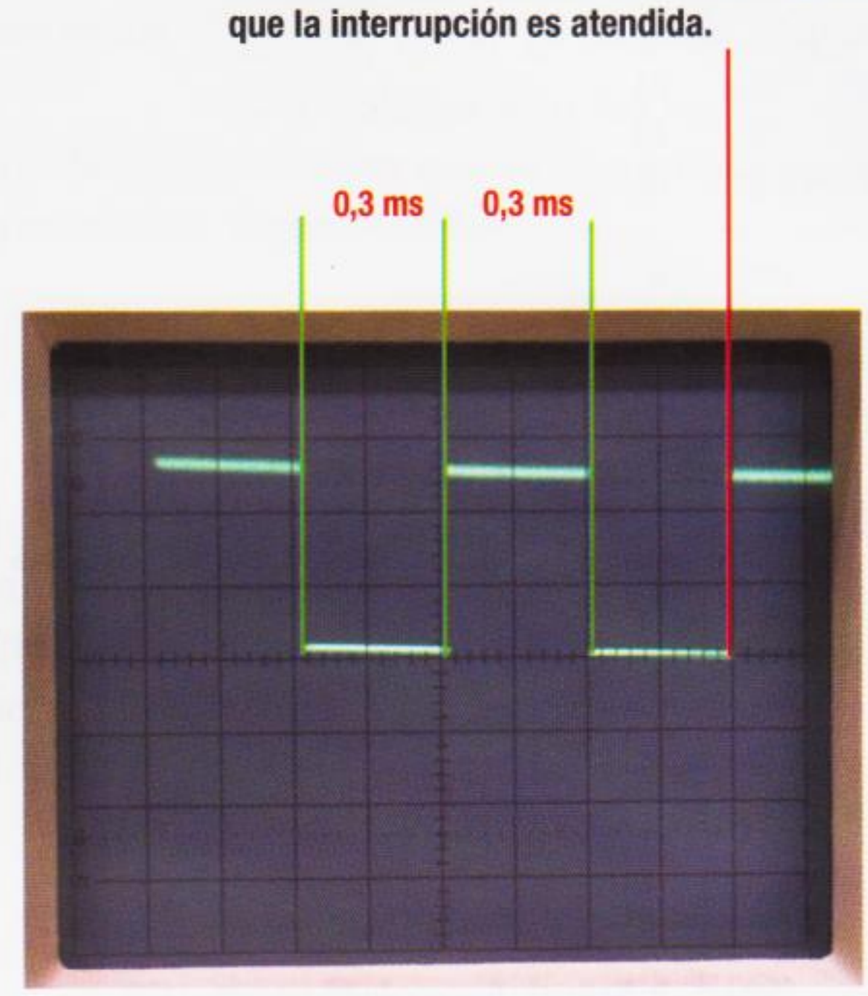


Figura 6.43 *Aquí se ha inicializado el TMR0 en 00 y se ha hecho un bypass del prescalador asignándolo al watchdog*

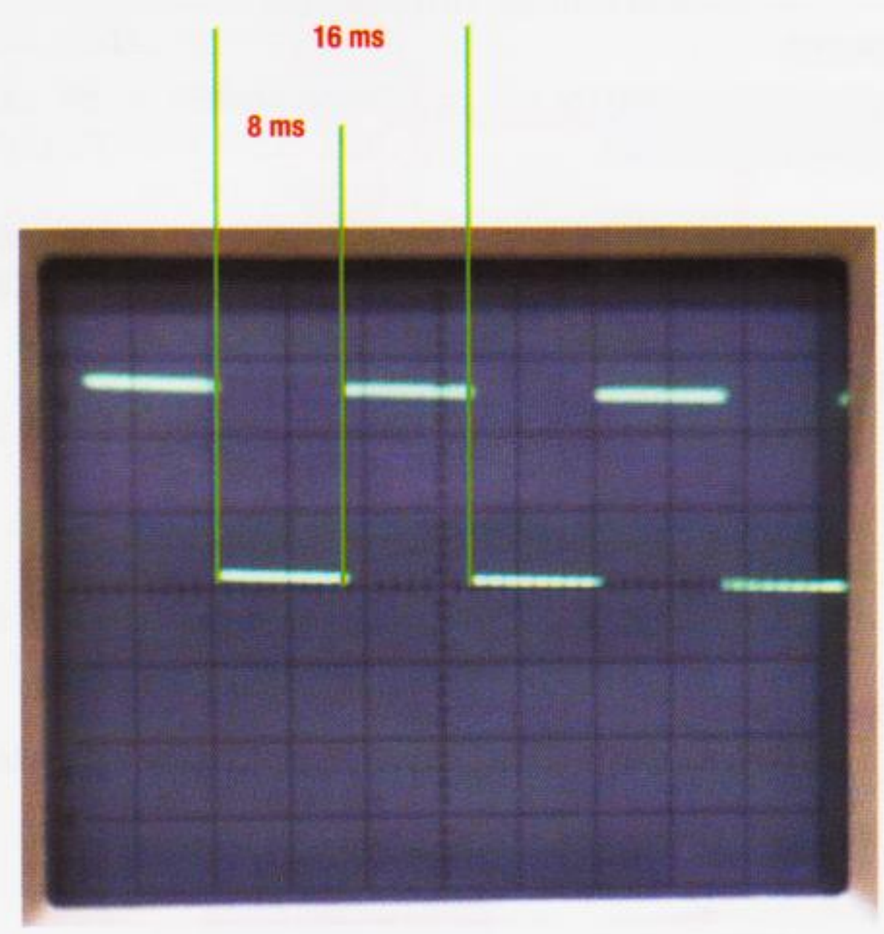
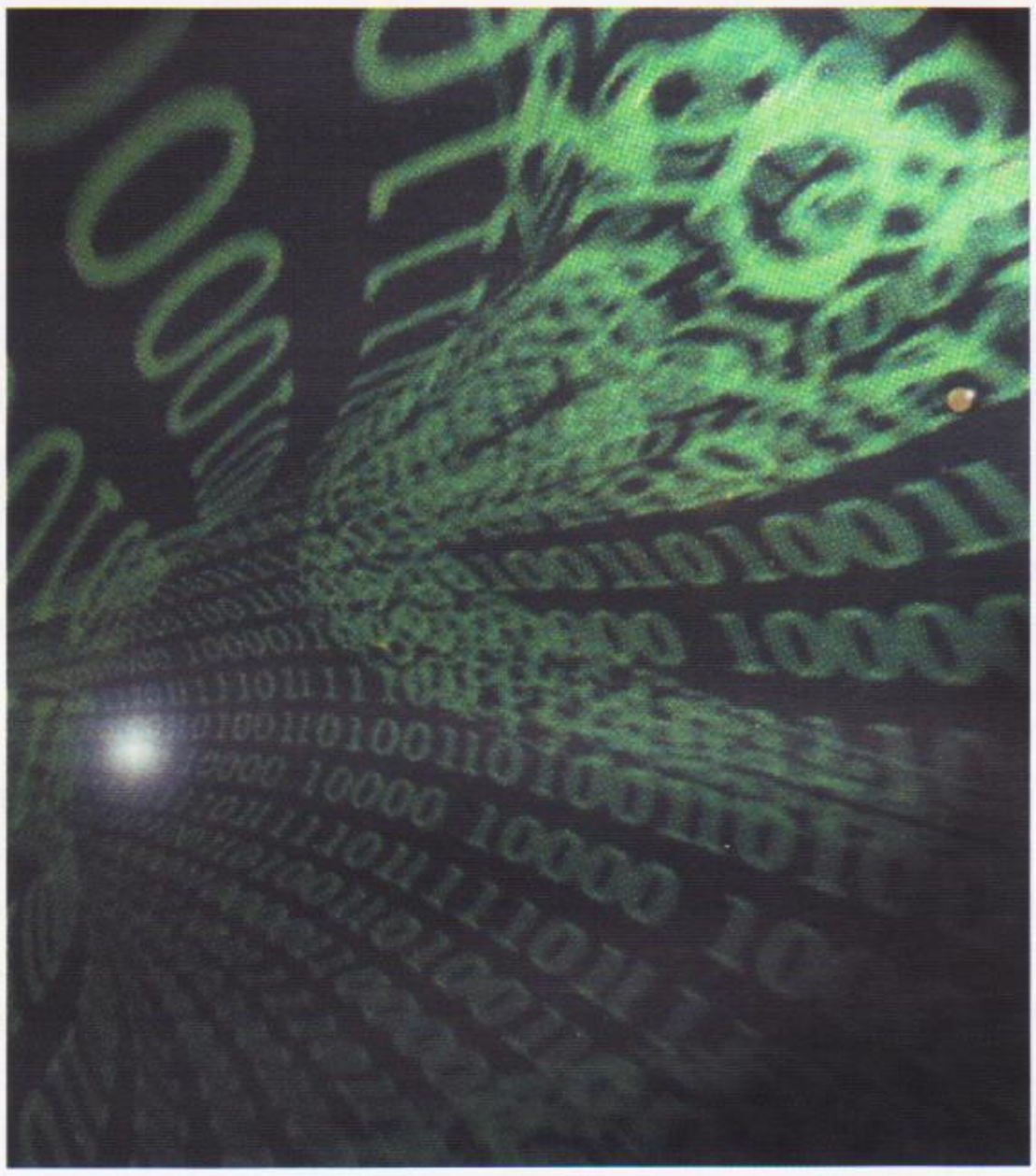


Figura 6.45 *Aquí se ha inicializado el TMR0 en 00 y la frecuencia de entrada se ha dividido por 32*

LECCIÓN 7



EJERCICIOS AVANZADOS DE PROGRAMACIÓN CON MICROCONTROLADORES PIC

Ya hemos conocido las principales herramientas para el desarrollo de proyectos con los microcontroladores PIC. También hemos estudiado algunos ejemplos que ilustran conceptos fundamentales para el manejo de estos dispositivos. Ahora estamos listos para entrar a analizar procesos más elaborados de programación como refuerzo a los conceptos expuestos en la sección de teoría.

Ejercicio No. 7
Conexión de memorias seriales al PIC

Las técnicas para almacenar información en medios electrónicos se perfeccionan más cada día. A diario vemos ejemplos de su utilización en nuestros hogares y oficinas, por ejemplo, en receptores de televisión, reproductores de discos compactos, sistemas de control remoto, impresoras, fotocopiadoras, teléfonos celulares, etc. Una de estas tecnologías corresponde a las llamadas memorias EEPROM seriales, las cuales tienen grandes ventajas si se comparan con otras posibilidades. Entre sus principales características se cuentan:

- Se pueden conectar fácilmente con microprocesadores o microcontroladores; inclusive algunos de ellos tienen pines dedicados para esta labor.
- Transferencia de datos de manera serial, lo que permite ahorrar pines del microcontrolador para dedicarlos a otras funciones.
- Ocupan la décima parte del espacio de las memorias que trabajan en paralelo. Esto permite ahorrar dinero debido al menor tamaño del circuito impreso.
- El consumo de corriente es mucho menor que en las memorias que trabajan en paralelo. Esto las hace ideales para sistemas portátiles que funcionan con baterías.

El objeto de este ejercicio es mostrar los aspectos más importantes de su tecnología y enseñar conceptos básicos para su utilización en circuitos reales. Se basa en las memorias que tienen comunicación a 2 hilos empleando la interfaz I²C, algunas de las más conocidas son 24LC01/02/04/16. La velocidad de transferencia de información para estos dispositivos es de 100 ó 400 kHz (aunque el límite lo impone el protocolo I²C, mas no la tecnología del dispositivo). Como característica importante de este elemento se tiene la inmunidad al ruido, dado que estos circuitos integrados tienen filtros en los pines de comunicación.

Estas memorias utilizan el bus de 2 hilos para comunicarse con otros dispositivos. Dado que cumplen con el protocolo I²C, tienen un pin llamado SCL que recibe los pulsos generados por el dispositivo maestro (o sea el microcontrolador) y otro llamado SDA que maneja el flujo de datos en forma bidireccional (entrada/salida). En la **figura 7.1** se muestra el diagrama de pines correspondiente a estas memorias.

Este dispositivo no requiere de un pin habilitador o *chip select*, ya que en este esquema la transferencia de información sólo se puede iniciar cuando el bus esté libre. En este caso, como cada dispositivo tiene su dirección determinada mediante los pines A0, A1 y A2, solamente responderá la memoria cuya dirección coincida con la dirección que va encabezando la trama de información. En la **figura 7.2** se muestra la capacidad de almacenamiento de estos dispositivos y las posibilidades de direccionamiento que tienen.

Transferencia de la información. Cuando el microcontrolador desea entablar comunicación con la memoria, debe enviarle una serie de *bits* que llevan la siguiente información:

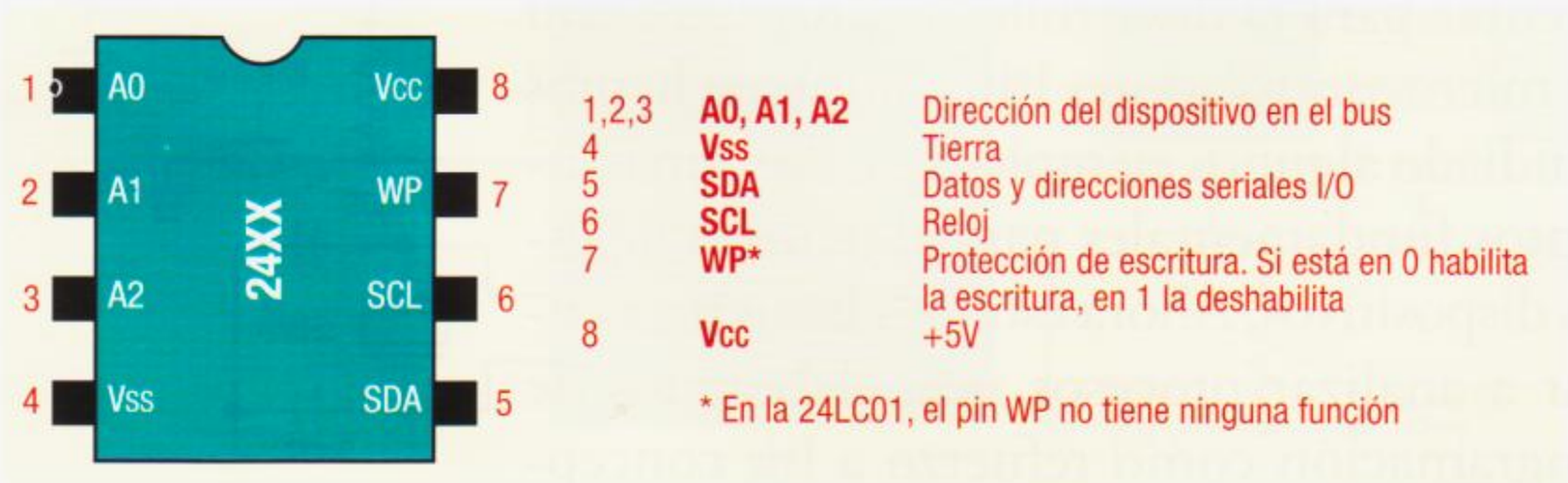


Figura 7.1 *Diagrama de pines de las memorias 24XX*

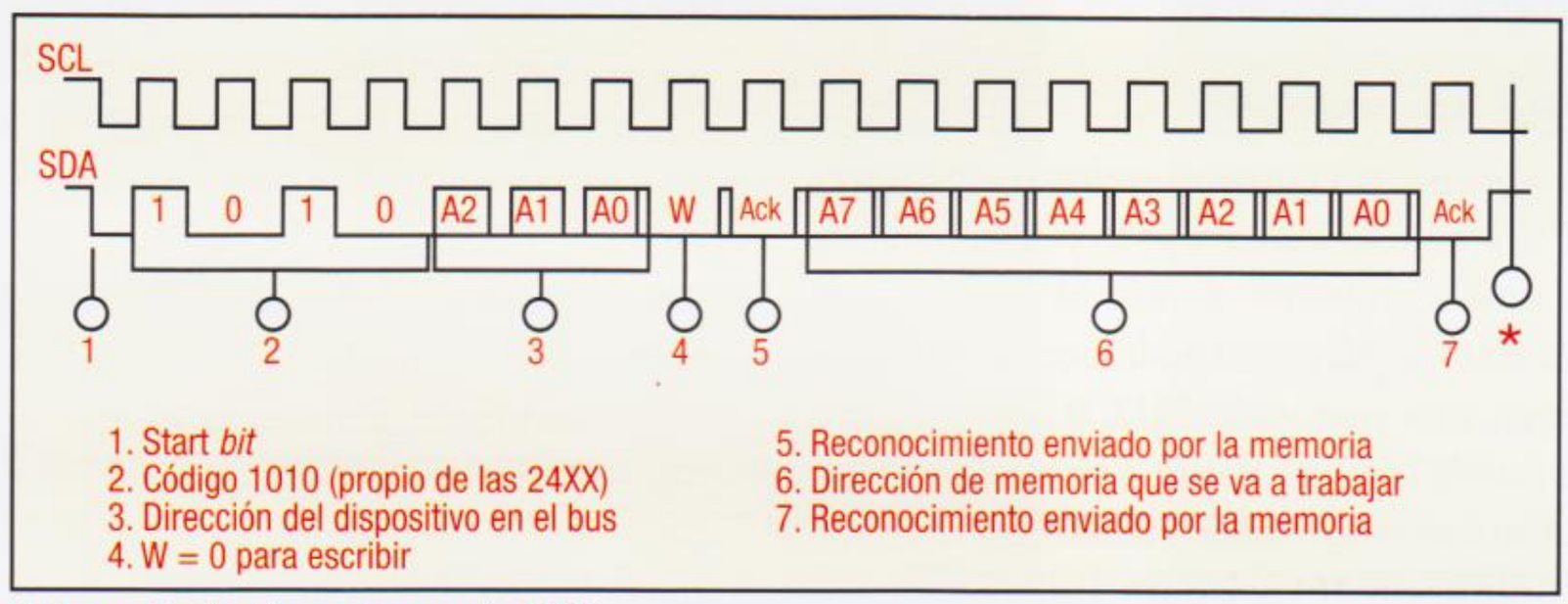
1. El *bit* de arranque o *start bit*
2. El código 1010 (propio de estas memorias)
3. La dirección del dispositivo (A2, A1, A0)
4. Un *bit* "0" el cual indica que se desea escribir en la memoria

Después de esto, la memoria debe enviar un reconocimiento para informarle al microcontrolador que recibió la información. Dicho asentimiento, llamado *ACK,* consiste en poner el bus en un nivel bajo (lo hace la memoria).

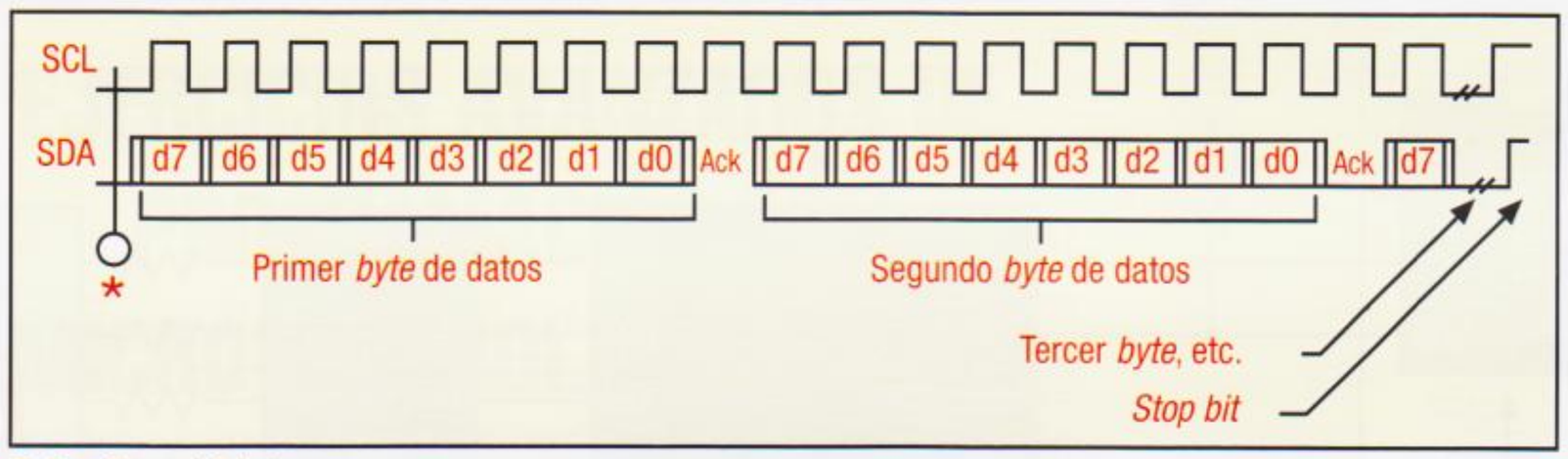
Referencia	Capacidad en Kbits	Bloques internos	A0	A1	A2	Dispositivos en el bus
24LC01B, 24C01	1	1	1 ó 0	1 ó 0	1 ó 0	8
24LC02B, 24C02	2	1	1 ó 0	1 ó 0	1 ó 0	8
24LC04B, 24C04	4	2	X	1 ó 0	1 ó 0	4
24LC08B	8	4	X	X	1 ó 0	2
24LC16B	16	8	X	X	X	1

Figura 7.2 *Capacidad de memoria y direccionamiento de las memorias 24LCXX*

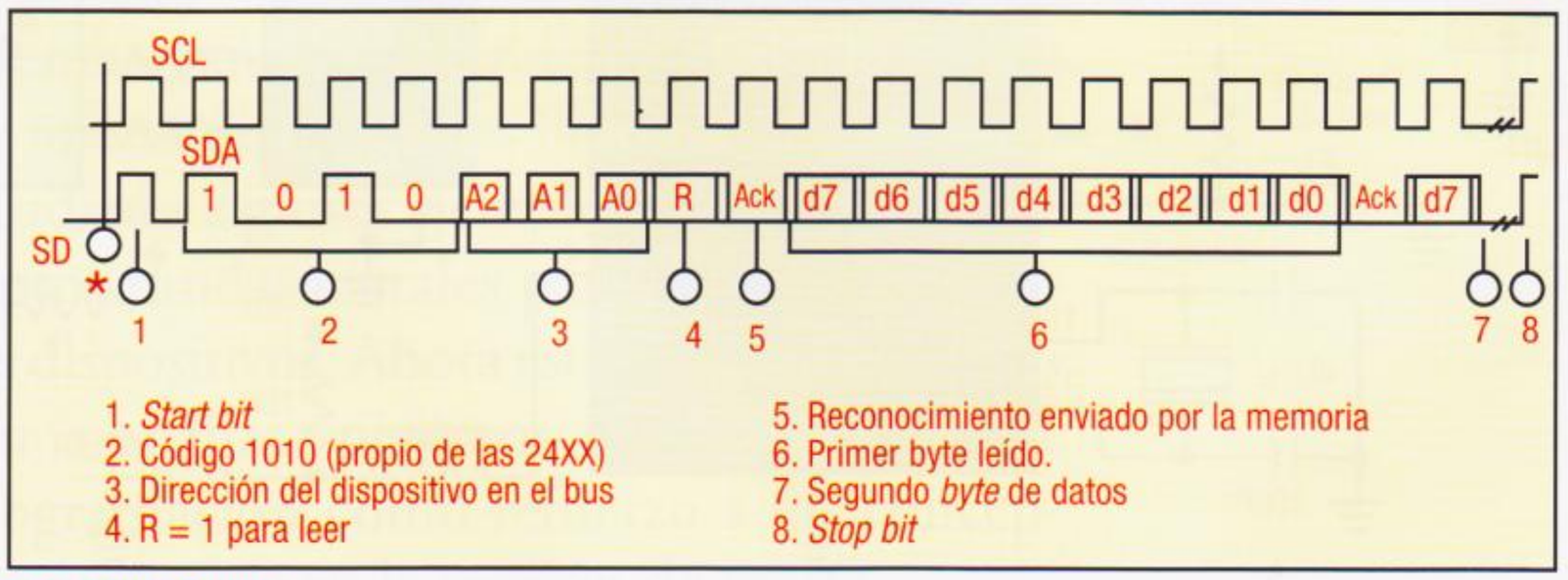
Luego, el microcontrolador debe enviar los *bits* que corresponden a la posición de memoria que se quiere leer o escribir; nuevamente la memoria envía un reconocimiento. El paso siguiente depende de la operación que se vaya a ejecutar.



A. Forma de direccionar la memoria 24XX



B. Escritura del *byte*



C. Lectura del *byte*

Figura 7.3 *Diagramas de tiempos para la lectura y la escritura en una memoria 24LCXX*

Si se trata de un proceso de escritura, el microcontrolador sólo debe enviar el dato que va a ser almacenado y esperar el asentimiento por parte de la memoria para confirmar que llegó correctamente. Si se trata de una lectura, nuevamente se deben repetir los primeros cuatro pasos, solo que en lugar de un "0", que indica escritura, se debe enviar un "1", que indica lectura. Después se espera el asentimiento y, acto seguido, se puede leer el byte con el dato que estaba en la posición de memoria que se indicó anteriormente. Cuando se termina la operación, el microcontrolador debe enviar una señal de parada (*stop bit*). En la **figura 7.3** se muestran los diagramas de tiempos correspondiente a todo el proceso descrito anteriormente.

Desarrollo del ejercicio

El ejercicio propuesto consiste en hacer un contador de 0 a 9 con un interruptor pulsador y *un display* de siete segmentos, donde el número que se muestra en el *display* se va a almacenar simultáneamente en una memoria 24LC01 (LC quiere decir que puede trabajar desde 2 voltios). Se va a utilizar un microcontrolador PIC16F84 (aunque se puede utilizar un 16C61 o un 16C71).

En la **figura 7.4** se muestra el diagrama esquemático del circuito. En este caso los pines de dirección de la memoria se conectaron a tierra, al igual que el pin WP. La resistencia de 4,7 kΩ conectada al pin SDA es necesaria dado que dicho pin tiene salida de colector abierto (*open collector*). El *display* se conecta al puerto A y el pulsador al pin RB0.

El programa que se escribe en el microcontrolador se muestra en la **figura 7.5**, su función principal es llevar el control de la cuenta decimal y almacenar en la memoria el mismo dato que se envía al *display*.

En el programa, la subrutina *WAIT* produce un retardo en milisegundos. La cantidad de milisegundos deseada debe escribirse en el registro *loops* antes de hacer el llamado correspondiente. Se utiliza principalmente para hacer un retardo de 10 milisegundos luego de escribir un dato en la memoria. Se debe tener en cuenta que los retardos están calculados para un oscilador de 4 MHz en el microcontrolador.

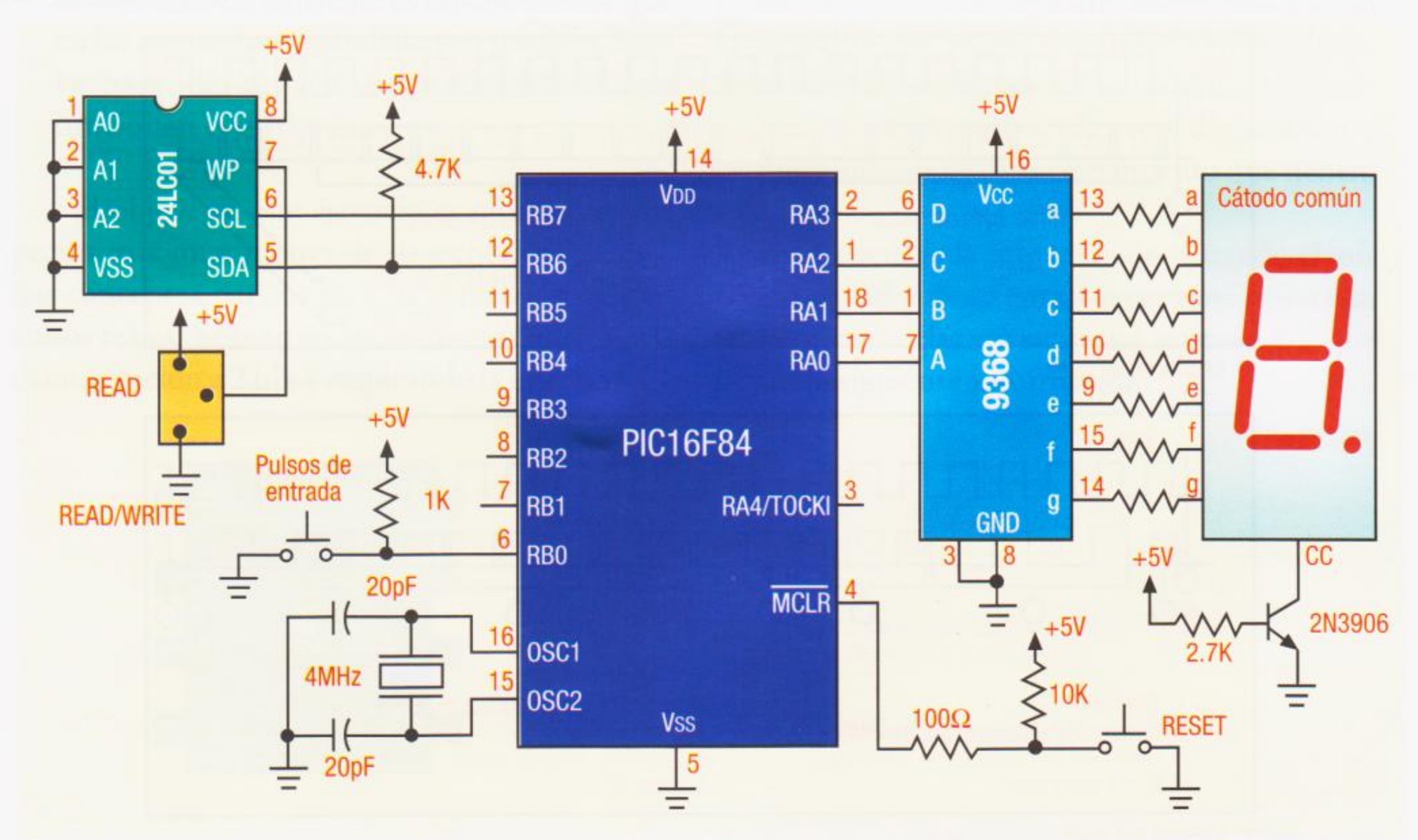


Figura 7.4 Diagrama esquemático del contador con PIC y memoria 24LC01

```
;ESTE PROGRAMA ELABORA UN CONTADOR DECIMAL CON UN
;PULSADOR Y UN DISPLAY DE 7 SEGMENTOS, EL VALOR DE
;LA CUENTA SE GUARDA EN LA MEMORIA SERIAL 24LC01
STATUS  EQU     3H          ;REGISTRO DE ESTADOS
PTOA    EQU     5H          ;
PTOB    EQU     6H          ;
ADDR    EQU    0DH          ;POSICIÓN DE MEMORIA QUE SE
                            ;LEE O ESCRIBE
DATAO   EQU    0EH          ;REGISTRO PARA ESCRIBIR
                            ;DATOS EN LA MEMORIA
SLAVE   EQU    0FH          ;DIRECCIÓN DEL DISPOSITIVO
                            ;EN EL BUS I2C (1010XXX0)
TXBUF   EQU    10H          ;BUFFER DE TRANSMISION
COUNT   EQU    11H          ;CONTADOR DE BITS
EEPROM  EQU    12H          ;BUFFER DE BITS
RXBUF   EQU    13H          ;BUFFER DE RECEPCIÓN
LOOPS   EQU    15H          ;SE UTILIZAN EN RETARDOS
LOOPS2  EQU    16H          ;
DI      EQU    7            ;BIT DE ENTRADA DESDE EEPROM
DO      EQU    6            ;BIT DE SALIDA PARA EEPROM
SDATA   EQU    6            ;LÍNEA DE DATOS SERIALES
                            ;(PIN RB6)
SCLK    EQU    7            ;RELOJ SERIAL (PIN RB7)
CONTA   EQU    17H          ;LLEVA LA CUENTA DE PULSACIONES
CONTA2  EQU    18H
TRISA   EQU    85H          ;REGISTRO DE CONFIGURACIÓN
                            ;DEL PUERTO A
TRISB   EQU    86H          ;REGISTRO DE CONFIGURACIÓN
                            ;DEL PUERTO B
Z       EQU    02H          ;BANDERA DE CERO DEL REGISTRO
                            ;DE ESTADOS
W       EQU    00H          ;INDICA QUE EL RESULTADO
                            ;SE GUARDA EN W
C       EQU    00H          ;BANDERA DE CARRY
        ORG     00H         ;VECTOR DE RESET
        GOTO    INICIO
        ORG     03H
WAIT                        ;SUBRUTINA DE RETARDO EN MILISEG.
TOP2    MOVLW   .110        ;EL NÚMERO DE MILISEGUNDOS LLEGA
        MOVWF   LOOPS2      ;CARGADO EN EL REGISTRO LOOPS
TOP     NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        DECFSZ  LOOPS2      ;PREGUNTA SI TERMINO 1 MS
        GOTO    TOP
        DECFSZ  LOOPS       ;PREGUNTA SI TERMINA EL RETARDO
        GOTO    TOP2
        RETLW   0
RETARDO MOVLW   .100        ;RETARDO DE 100 MILISEGUNDOS
        MOVWF   LOOPS
        CALL    WAIT
        RETLW   0
BSTART ;ESTA RUTINA GENERA EL START BIT PARA LA COMUNICACIÓN SERIAL
        MOVLW   B'00111111'
        TRIS    PTOB        ;PROGRAMAR DATOS Y RELOJ
                            ;COMO SALIDAS
        BCF     PTOB,SCLK   ;LÍNEA DE RELOJ EN NIVEL BAJO
        BSF     PTOB,SDATA  ;SE ASEGURA LÍNEA DE DATOS EN ALTO
        NOP
        BSF     PTOB,SCLK   ;LÍNEA DE RELOJ EN ALTO
        NOP
        NOP                 ;AJUSTE DE TIEMPO
        NOP
        NOP
        NOP
        BCF     PTOB,SDATA  ;SE BAJA LA LÍNEA DE DATOS
        NOP                 ;MIENTRAS EL RELOJ ESTÁ ALTO
        NOP                 ;AJUSTE DE TIEMPO
        NOP
```

```
        NOP
        BCF     PTOB,SCLK   ;SE BAJA LA LÍNEA DE RELOJ
        NOP                 ;PARA TERMINAR EL PULSO
        NOP
        RETLW   0
BSTOP                       ;ESTA RUTINA GENERA EL STOP BIT
                            ;PARA LA COMUNICACIÓN SERIAL
        MOVLW   B'00111111'
        TRIS    PTOB        ;PROGRAMA RELOJ Y DATOS
                            ;COMO SALIDAS
        BCF     PTOB,SDATA  ;ASEGURA LÍNEA DE DATOS EN BAJO
        NOP
        NOP
        NOP
        BSF     PTOB,SCLK   ;LÍNEA DE RELOJ EN NIVEL ALTO
        NOP
        NOP
        NOP
        BSF     PTOB,SDATA  ;LA LÍNEA DE DATOS PASA A  ALTO
        NOP                 ;MIENTRAS EL RELOJ ESTÁ ALTO
        NOP
        BCF     PTOB,SCLK   ;LA LÍNEA DE RELOJ BAJA
        NOP                 ;NUEVAMENTE
        NOP                 ;PARA COMPLETAR EL PULSO
        NOP
        RETLW   0
BITOUT ;ESTA RUTINA TOMA EL BIT QUE SE DEBE
       ;TRANSMITIR Y LO SACA AL PUERTO
        MOVLW   B'00111111';ADEMÁS GENERA EL PULSO
                            ;DE RELOJ
        TRIS    PTOB        ;PROGRAMA RELOJ Y DATOS
                            ;COMO SALIDAS
        BSF     PTOB,SDATA  ;ASUME QUE EL BIT ES ALTO
        BTFSS   EEPROM,DO   ;PREGUNTA ESTADO DEL BIT
                            ;QUE SE VA A TRANSMITIR
        BCF     PTOB,SDATA  ;SI EL BIT ES BAJO PONE
                            ;LA SALIDA EN BAJO
CLKOUT  NOP
        NOP
        BSF     PTOB,SCLK   ;SUBE EL NIVEL DE LA
        NOP                 ;LÍNEA DE RELOJ
        NOP                 ;PARA FORMAR EL PULSO
        NOP
        NOP
        BCF     PTOB,SCLK   ;TERMINA PULSO DE RELOJ
        RETLW   0
BITIN  ;ESTA RUTINA LEE UN BIT DE LA MEMORIA Y LO PONE EN
       ;UN REGISTRO
        BSF     EEPROM,DI   ;ASUME QUE EL BIT ES DE
                            ;NIVEL ALTO
        MOVLW   B'01111111' ;PROGRAMA PIN DE DATOS
                            ;COMO ENTRADA
        TRIS    PTOB
        BSF     PTOB,SCLK   ;SUBE LA LINEA DEL RELOJ
        NOP
        NOP
        NOP
        NOP
        NOP                 ;
        BTFSS   PTOB,SDATA  ;¿ESTADO DEL PIN DE DATOS?
        BCF     EEPROM,DI   ;SI ES BAJO LO PONE EN ESE
NIVEL   BCF     PTOB,SCLK   ;SI ES ALTO LO DEJA COMO
        RETLW   0           ;SE ASUMIÓ ANTES
TX     ;ESTA RUTINA SE ENCARGA DE TRANSMITIR UN BYTE
       ;HACIA LA MEMORIA
        MOVLW   .8
        MOVWF   COUNT       ;EL NÚMERO DE BITS ES 8
TXLP    BCF     EEPROM,DO   ;ASUME QUE EL BIT A ENVIAR ES BAJO
        BTFSC   TXBUF,7     ;CONSULTA EL ESTADO REAL DEL BIT
        BSF     EEPROM,DO   ;SI ERA ALTO LO DEJA CON DICHO NIVEL
        CALL    BITOUT      ;SACA EL BIT POR EL PUERTO
        RLF     TXBUF,1     ;ROTA EL BYTE QUE SE ESTÁ
```

```
                                ;TRANSMITIENDO
        DECFSZ  COUNT           ;PREGUNTA SI YA PASARON
                                ;LOS 8 BITS
        GOTO    TXLP            ;SI NO HA TERMINADO SIGUE
                                ;TRANSMITIENDO
        CALL    BITIN           ;ESPERA EL RECONOCIMIENTO
                                ;ENVIADO POR LA
        RETLW   0               ;MEMORIA (ACK)
RX      ;ESTA RUTINA RECIBE UN BYTE Y LO ENTREGA EN
         ;EL REGISTRO RXBUF
        CLRF    RXBUF           ;BORRA EL BUFFER DE ENTRADA
        MOVLW   .8              ;INDICA QUE RECIBE 8 BITS
        MOVWF   COUNT
        BCF     STATUS,0        ;BORRA EL CARRY
RXLP    RLF     RXBUF, F        ;ROTA A LA IZQUIERDA
        CALL    BITIN           ;LEE UN BIT
        BTFSC   EEPROM,DI
        BSF     RXBUF,0         ;SI ES NECESARIO PONE EL BIT, EN UNO
        DECFSZ  COUNT           ;PREGUNTA SI COMPLETO 8 BITS
        GOTO    RXLP            ;SI NO, RECIBE OTRO BIT
        BSF     EEPROM,DO       ;ENVIA EL ACK DE
ASENTIMIENTO
        CALL    BITOUT          ;PARA TERMINAR
        RETLW   0
LEER    ;ESTA RUTINA RECIBE LA DIRECCIÓN QUE SE
       ;DESEA LEER Y DEVUELVE EL DATO QUE TIENE GRABADO
        CALL    BSTART          ;GENERA EL START BIT
        NOP                     ;
        NOP                     ;
        BCF     SLAVE,0         ;SELECCIONA LA MEMORIA
        MOVF    SLAVE,W         ;Y SELECCIONA MODO DE
                                ;ESCRITURA
        MOVWF   TXBUF           ;
        CALL    TX              ;ENVÍA ESOS DATOS A LA
MEMORIA
        MOVF    ADDR,W          ;
        MOVWF   TXBUF           ;ENVÍA LA POSICIÓN DE
                                ;MEMORIA QUE VA A SER LEIDA
        CALL    TX              ;
        NOP                     ;AHORA SE SELECCIONA
                                ;NUEVAMENTE LA MEMORIA
        NOP                     ;Y SE LE INDICA MODO DE LECTURA
        CALL    BSTART          ; GENERA START BIT
        NOP
        NOP
        BSF     SLAVE,0         ;INDICA QUE SE VA A LEER
        MOVF    SLAVE,W         ;SELECCIONA EL DISPOSITIVO
        MOVWF   TXBUF           ;
        CALL    TX              ;ENVÍA ESA INFORMACIÓN
        NOP                     ; A LA MEMORIA.
        CALL    RX              ;LA MEMORIA ENTREGA EL
                                ;BYTE DE ESA DIRECCIÓN
        BSF     EEPROM,DO       ;ENVIA EL ACK DE
                                ;RECONOCIMIENTO
        CALL    BITOUT
        CALL    BSTOP           ;SE ENVÍA EL STOP BIT PARA
         RETLW   0              ;FINALIZAR COMUNICACIÓN
ESCRIB  ;ESTA RUTINA ESCRIBE UN DATO EN LA POSICIÓN
        ;DE MEMORIA QUE SE LE INDIQUE EN EL REGISTRO ADDR
        CALL    BSTART          ;GENERA EL START BIT
        NOP
        NOP
        BCF     SLAVE,0         ;SELECCIONA LA MEMORIA
        MOVF    SLAVE,W         ;Y SELECCIONA MODO DE ESCRITURA
        MOVWF   TXBUF
        CALL    TX              ;ENVÍA ESOS DATOS A LA MEMORIA
        MOVF    ADDR,W
        MOVWF   TXBUF           ;ENVÍA LA POSICIÓN DE
                                ;MEMORIA QUE VA A SER GRABADA
        CALL    TX              ;AHORA SE SELECCIONA
```

```
        NOP                     ;NUEVAMENTE LA MEMORIA
        NOP                       ;Y SE LE INDICA MODO DE LECTURA
        MOVF    DATAO,W         ;TOMA EL DATO QUE VA A
        MOVWF   TXBUF           ; SER GRABADO Y LO ENVÍA
        CALL    TX
        CALL    BSTOP
        MOVLW   .10             ;RETARDO DE 10 MS AL ESCRIBIR
        MOVWF   LOOPS           ;CADA DATO
        CALL    WAIT
        RETLW   0
INICIO  BSF     STATUS,5        ;SE UBICA EN EL SEGUNDO
                                ;BANCO DE RAM
         MOVLW     0F0H         ;SE CARGA EL REGISTRO W CON 0F
         MOVWF     TRISA        ;SE PROGRAMA EL PUERTO A
                                ;COMO SALIDAS
         MOVLW     07FH         ;SE CARGA EL REGISTRO W CON 00
         MOVWF     TRISB        ;SE PROGRAMA EL PUERTO B
                                ;COMO ENTRADAS
         BCF       STATUS,5 ;SE UBICA EN EL PRIMER
                                ;BANCO DE MEMORIA RAM
         MOVLW  B'10100000';LA DIRECCIÓN A0, A1, Y
                                ;A2 DE LA MEMORIA
         MOVWF     SLAVE        ;EN EL BUS I2C ES 000
          CLRF      ADDR        ;CUANDO SE ENCIENDE EL
                                ;SISTEMA SE VERIFICA QUE
         CALL      LEER         ;EL DATO GUARDADO EN
                                ;MEMORIA ESTÁ ENTRE 0 Y 9
         MOVLW     0AH          ;LA PRUEBA SE HACE PORQUE
                                ;LA PRIMERA VEZ QUE
         SUBWF     RXBUF,W;SE ENCIENDA EL SISTEMA
                                ;SE PUEDE TENER UN
         BTFSS     STATUS,C  ;NÚMERO FUERA DEL RANGO
         GOTO      CICLO        ;PARA LAS OCASIONES
                                ;POSTERIORES NO IMPORTA
INI2     CLRF      CONTA        ;INICIA CONTADOR EN CERO
         CLRF      DATAO
         CALL      ESCRIB ;INICIA DATO DE MEMORIA EN 0
CICLO    CALL      LEER         ;LEER MEMORIA, DEVUELVE
                                ;DATO EN W
         MOVF      RXBUF,W;PASA EL VALOR DE W AL
                                ;PUERTO A (DISPLAY)
         MOVWF     CONTA
         MOVWF     PTOA
         CALL      RETARDO;RETARDO ESPERANDO QUE
                                ;SUELTEN LA TECLA
PULSA    BTFSC     PTOB,0 ;PREGUNTA SI EL PULSADOR
                                ;ESTÁ OPRIMIDO
         GOTO      PULSA        ;SI NO LO ESTÁ CONTINÚA
                                ;REVISÁNDOLO
         CALL      RETARDO;SI ESTÁ OPRIMIDO RETARDA
                                ;100 MILISEGUNDOS
         BTFSC     PTOB,0 ;PARA COMPROBAR
         GOTO      PULSA        ;SI NO LO ESTÁ VUELVE A REVISAR
         INCF      CONTA        ;SI LO CONFIRMA
                                ;INCREMENTA EL CONTADOR
         MOVF      CONTA,W;CARGA EL REGISTRO W CON
                                ;EL VALOR DE LA CUENTA
         MOVWF     DATAO        ;EL DATO DE LA CUENTA LO
                                ;GUARDA EN MEMORIA
         CALL      ESCRIB ;PARA RECUPERARLO EN CASO
                                ;DE UN CORTE DE ENERGÍA
         MOVF      CONTA,W
         XORLW     0AH          ;HACE OPERACIÓN XOR PARA
                                ;VER SI ES IGUAL A 0AH
         BTFSS     STATUS,Z  ;PRUEBA SI EL CONTADOR
                                ;LLEGÓ A 0AH (DIEZ)
         GOTO      CICLO        ;SI NO ES IGUAL SE
                                ;INCREMENTA NORMALMENTE
         GOTO      INI2
         END
```

Figura 7.5 *Programa para el manejo de una memoria serial*

La subrutina *BSTART* genera el *bit* de inicio de la comunicación, con la temporización y estado de los pines adecuados. Por su parte, la subrutina *BSTOP* hace lo mismo con el *bit* de parada o de fin de la comunicación. La subrutina *BITOUT* toma el *bit* del dato que se debe transmitir y lo envía hacia la memoria, encargándose de programar el pin del microcontrolador como salida y de generar el pulso de reloj necesario para la sincronización. La rutina *BITIN* hace su parte cuando se está leyendo un *bit* enviado por la memoria, genera el pulso de reloj y pasa el *bit* leído al registro o *buffer* de entrada. Las rutinas *TX* y *RX* se encargan de transmitir y recibir un *byte* completo de datos. Cada una hace ocho llamados seguidos a las rutinas *BITOUT* y *BITIN* respectivamente.

La rutina *LEER* recibe en el registro *addr* la posición de memoria que se debe leer y genera todas las señales necesarias (incluyendo el *bit* de *start* y el *bit* de *stop*) para obtener el dato que en ella se encuentra grabado. Al final devuelve el dato que recibió de la memoria en el registro *rxbuf*. La rutina *ESCRIB* toma el dato contenido en el registro *DATAO* y lo escribe en la posición de la memoria que está direccionada en el registro *addr*.

Cada vez que se enciende el sistema, el microcontrolador lee el dato que se encuentra en la primera posición de memoria y lo pasa al *display*. Una vez que el pulsador sea oprimido, se debe incrementar dicho dato y actualizarse el *display*, al tiempo que se vuelve a almacenar ese número en la memoria. Un caso especial ocurre cuando se enciende el sistema por primera vez. Como el dato que se encuentra grabado en la memoria es desconocido, y podría ser superior a 9, éste se debe probar, y si se encuentra que es mayor, se borra y se empieza la cuenta en 0.

Las rutinas que permiten leer y escribir en la memoria 24LC01se pueden utilizar como parte de cualquier programa sin que se tengan contratiempos. Sólo se debe tener en cuenta que las temporizaciones están calculadas para un oscilador de 4 MHz. Con las rutinas LEER y ESCRIB se tiene una velocidad de transferencia de información de aproximadamente 60 kHz.

Una prueba, que es muy interesante, consiste en cambiar de posición el interruptor que selecciona la protección de escritura. Cuando está en la posición READ/WRITE, se puede incrementar el contador normalmente, mientras que cuando se encuentra en la posición READ el contador no se incrementa debido a que la memoria está protegida contra escritura. A continuación detallamos tres instrucciones utilizadas en los ejercicios anteriores.

CALL. Llama una subrutina que está ubicada en la posición de memoria o etiqueta "k".
Sintaxis: CALL k
Ejemplo: **aquí** CALL **rutina**
Antes de la instrucción: contador de programa = **aquí**
Después de la instrucción: contador de programa = **rutina**
Pila = dirección **aquí**

RLF. Rote el contenido del registro "f" a la izquierda, usando el *carry*.
Sintaxis: RLF f,d ;si d= 0 el resultado se ;guarda en W, si d=1 el ;resultado se guarda en f
Ejemplo: RLF **regis**,0
Antes de la instrucción: C=0, W=?, **regis** = 11100110
Después de la instrucción: C=1, W=11001100
regis = 11100110

DECFSZ. Decrementa el contenido del registro "f"; si el contenido queda en 00, salta una línea.

Sintaxis: DECFSZ f,d
Ejemplo: **aquí** DECFSZ **regis**,1
GOTO ciclo
Continúa....
Antes de la instrucción: contador de programa = aquí
Después de la instrucción: **regis** = regis – 1
si **regis** = 0, entonces
contador de programa = continúa
si **regis** <> 0, entonces
contador de programa = **aquí** + 1

Ejercicio No. 8
Manejo de un módulo LCD

Cuando se trabaja en el diseño de circuitos electrónicos es frecuente encontrar la necesidad de visualizar un mensaje que tiene que ver con el estado de la máquina que se va a controlar, con instrucciones para el operario; o si es un instrumento de medida, mostrar el valor registrado. En la mayoría de los casos recurrimos a los *displays* de siete segmentos; pero éstos, además de no mostrar caracteres alfanuméricos ni ASCII, tienen un elevado consumo de corriente y son un poco dispendiosos de manejar, cuando se requiere hacer multiplexaje.

Los módulos de cristal líquido o LCD solucionan estos inconvenientes y presentan algunas ventajas, como un menor consumo de corriente, no hay que preocuparse por hacer multiplexaje, no hay que hacer tablas especiales con los caracteres que se desea mostrar, se pueden conectar fácilmente con microprocesadores o microcontroladores y, además, los proyectos adquieren una óptima presentación y funcionalidad. En principio, vamos a conocer las características más importantes de los módulos, luego mostramos la forma de conectarlos con el microcontrolador y presentaremos programas simples para escribir mensajes en la pantalla.

Módulos de cristal líquido o LCD

Antes de mostrar la forma de conectar estos módulos con el microcontrolador, haremos un pequeño recuento de sus principales características, las cuales nos servirán para entender mejor los programas y los diagramas queveremos más adelante:

- Los módulos LCD se encuentran en diferentes presentaciones, por ejemplo (2 líneas por 16 caracteres), 2x20, 4x20, 4x40, etc. La forma de utilizarlos y sus interfaces son similares, por eso, los conceptos vistos aquí se pueden emplear en cualquiera de ellos. En nuestro caso trabajaremos con un *display* de 2x16, ya que es de bajo costo, se consigue fácilmente y tiene un tamaño suficiente para la mayoría de los circuitos.

- La **figura** 7.6 contiene una de las configuraciones de pines más comunes en que se encuentran estos dispositivos. Algunos módulos LCD tienen una luz posterior o *backlight*, para mejorar su visualización; ésta se maneja a través de dos pines que normalmente se conectan a +5V y a tierra. Para evitar que se presenten altas temperaturas debido a la luz posterior, estos pines los debemos manejar de manera pulsante (encendiendo y apagando), con una frecuencia de aproximadamente 60 Hz. Otra opción mucho más sencilla es utilizar una resistencia de 10 ohmios (a 1/2W) para alimentar el terminal positivo de esta luz posterior.

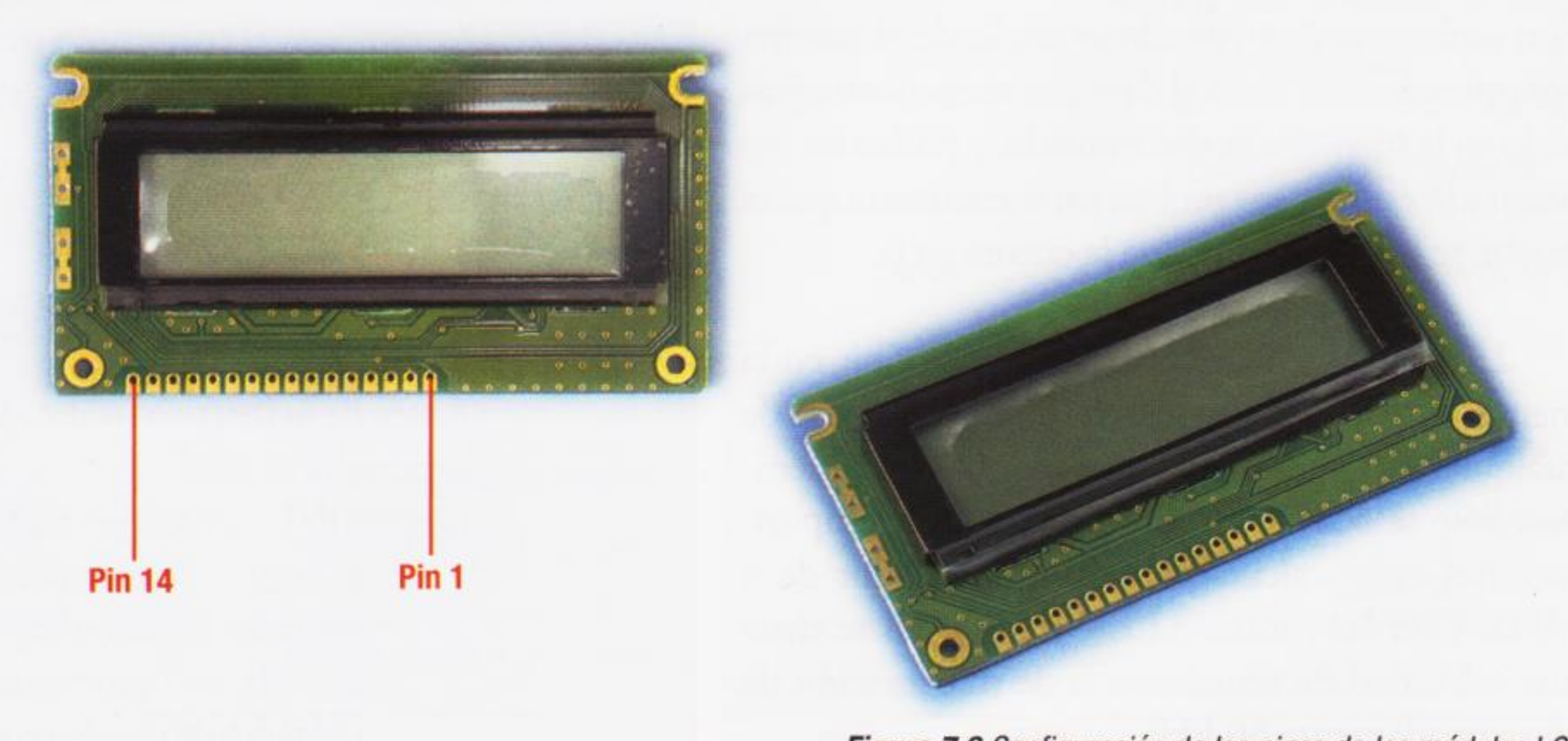


Figura 7.6 *Configuración de los pines de los módulos LCD*

Terminal	Símbolo	Nombre y función
1	Vss	Tierra, 0V
2	Vdd	Alimentación +5V
3	Vo	Ajuste de voltaje de contraste
4	RS	Selección dato/control
5	R/W	Lectura/escritura en LCD
6	E	Habilitación
7	D0	D0 Bit menos significativo
8	D1	D1
9	D2	D2
10	D3	D3
11	D4	D4
12	D5	D5
13	D6	D6
14	D7	D7 *Bit* más significativo

Figura 7.7 *Función de los pines del módulo LCD*

- Los pines de conexión de estos módulos incluyen un bus de datos de 8 *bits*, un pin de habilitación (E), un pin de selección que nos indica que el dato es una instrucción o un carácter del mensaje (RS), y un pin que indica si se va a escribir o leer en el módulo LCD (R/W). La **figura** 7.7 describe la función de cada uno de ellos.

- Según la operación que deseemos hacer sobre el módulo de cristal líquido, los pines de control E, RS y R/W deben tener un estado determinado. Además, debe tener en el bus de datos un código que indique un carácter para mostrar en la pantalla o una instrucción de control. En la **figura** 7.8 se muestra el diagrama de tiempos que debemos cumplir para manejar el módulo.

- El módulo LCD responde a un conjunto especial de instrucciones; éstas deben ser enviadas por el microcontrolador o sistema de control al *display*, según la operación que se requiera. Estas instrucciones se emplean en los ejemplos con los cuales practicaremos más adelante, en ellos se explica la forma de utilizarlas. En la **figura** 7.9 se muestran las instrucciones del módulo.

- La interfaz entre el microcontrolador y el *display* de cristal líquido se puede hacer con el bus de datos trabajando a 4 u 8 *bits*. Las señales de control trabajan de la misma forma en cualquiera de los dos casos; la diferencia se establece en el momento de inicializar el sistema, ya que existe una instrucción que permite establecer dicha configuración. Estas conexiones se explican más adelante en forma detallada.

- Los caracteres que se envían al *display* se almacenan en la memoria RAM del módulo. Existen posiciones de memoria RAM cuyos datos son visibles en la pantalla y otras que no son visibles; estas últimas se pueden utilizar para guardar caracteres que luego se desplazan hacia la parte visible. En la **figura** 7.10 se muestran las direcciones de memoria visibles y no visibles, que conforman las dos líneas de caracteres del módulo.

- Es importante anotar que sólo se pueden mostrar caracteres ASCII de 7 *bits*, por lo tanto algunos caracteres especiales no se pueden ver

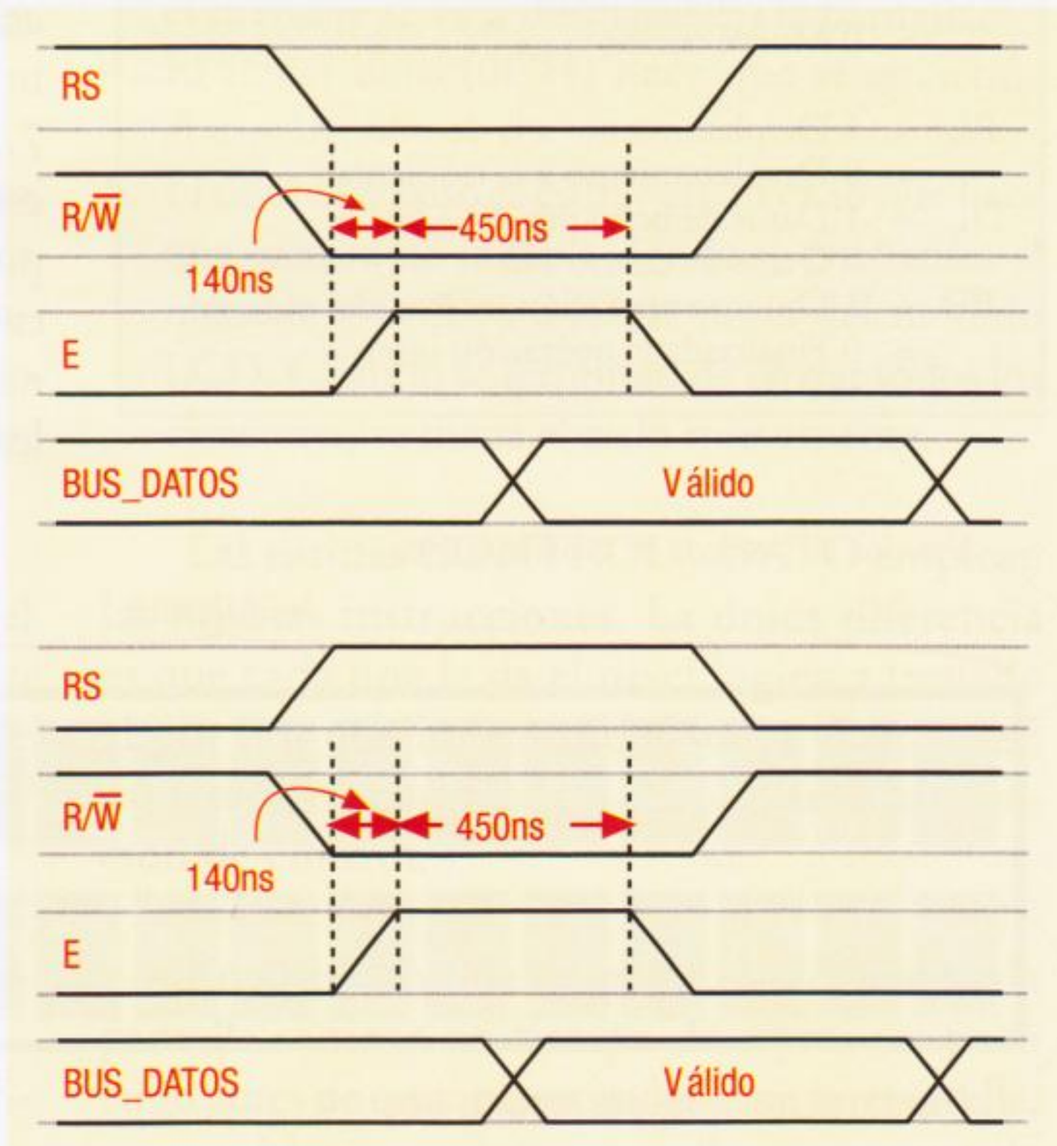


Figura 7.8 *Diagrama de tiempos del módulo LCD*

Control y dato / INSTRUCCIONES	Señal de control		DATO/DIRECCIÓN								DESCRIPCIÓN
	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0	
Borrar pantalla	0	0	0	0	0	0	0	0	0	1	Limpia todo el *display* y retorna el cursor a la posición de inicio (dirección 0)
Cursor a casa	0	0	0	0	0	0	0	0	1	*	Retorna el cursor a la posición de inicio (dirección 0). También retorna el display, desplazándolo a la posición original. Los contenidos de la RAM DD permanecen sin cambio.
Seleccionar modo	0	0	0	0	0	0	0	1	1/D	0	Configura la dirección de movimiento del cursor y si se desplaza o no el *display*. Esta operación es ejecutada durante operaciones de lectura y escritura.
Encender/apagar pantalla	0	0	0	0	0	0	1	D	C	B	Configura el estado ON/OFF de todo el *display* (D), el cursor (C) y el parpadeo del carácter en la posición del cursor.
Desplazar cursor/pantalla	0	0	0	0	0	1	S/C	R/L	*	*	Mueve el cursor y desplaza el *display* sin cambiar los contenidos de la RAM DD.
Acticar función	0	0	0	0	1	D/L	N	F	*	*	Configura el tamaño de la interfaz (DL), el número de líneas del *display* (N) y la fuente del carácter (F). N=0, 1 línea N=1, 2 líneas
CG RAM	0	0	0	1	Dirección generador de RAM						Ajusta la dirección del generador de caracteres. El dato CG RAM es enviado y recibido después de este ajuste.
DD RAM	0	0	1	Direcciónes de datos RAM							Ajusta la dirección de la RAM DD. La dirección es enviada y recibida después de este ajuste.
Bandera de ocupado	0	0	BF	AC							Lectura de la bandera *Busy Flag*, indicando que operaciones internas son efectuadas, y lectura de los contenidos del contador de direcciones.
Escritura CG RAM/DD RAM	1	0	Escritura de dato								Escribe datos en la RAM DD o en la RAM CG.
LECTURA CG RAM/DD RAM	1	1	Lectura de dato								Lectura de datos desde la RAM DD o la RAM CG

Figura 7.9 *Instrucciones DEL MÓDULO LCD*

Significado de las abreviaturas

I/D = 1 Incrementa
= 0 Decrementa
S = 1 Desplaza el mensaje en la pantalla
= 0 Mensaje fijo en la pantalla
D = 1 Encender (activar) la pantalla
= 0 Apagar la pantalla (desactivar)

C = 1 Activar cursor
= 0 Desactivar cursor
B = 1 Parpadea carácter señalado por el cursor
= 0 No parpadea el carácter
S/C = 1 Desplaza pantalla
= 0 Mueve cursor

RL = 1 Desplazamiento a la derecha
= 0 Desplazamiento a la izquierda
DL = 1 Datos de ocho *bits*
= 0 Datos de cuatro *bits*
BF = 1 Durante operación interna del módulo
= 0 Finalizada la operación interna

(debemos tener a mano una tabla de los caracteres ASCII para conocer los datos que son prohibidos). Por otra parte, tenemos la opción de crear caracteres especiales (creados por el programador), y almacenarlos en la memoria RAM que posee el módulo.

Interfaz con microcontrolador a 8 *bits*

En este ejercicio vamos a conectar el módulo de cristal líquido a un microcontrolador PIC16F84, utilizando el bus de datos a 8 *bits*. En la **figura 7.11** se muestra el diagrama de conexiones para este caso. Cabe anotar que los módulos LCD se pueden conectar a una gran variedad de microcontroladores y en cada caso se deben acondicionar las diferentes conexiones (bus de datos, señales de control, etc.).

Figura 7.10 *Mapa de memoria del módulo LCD*

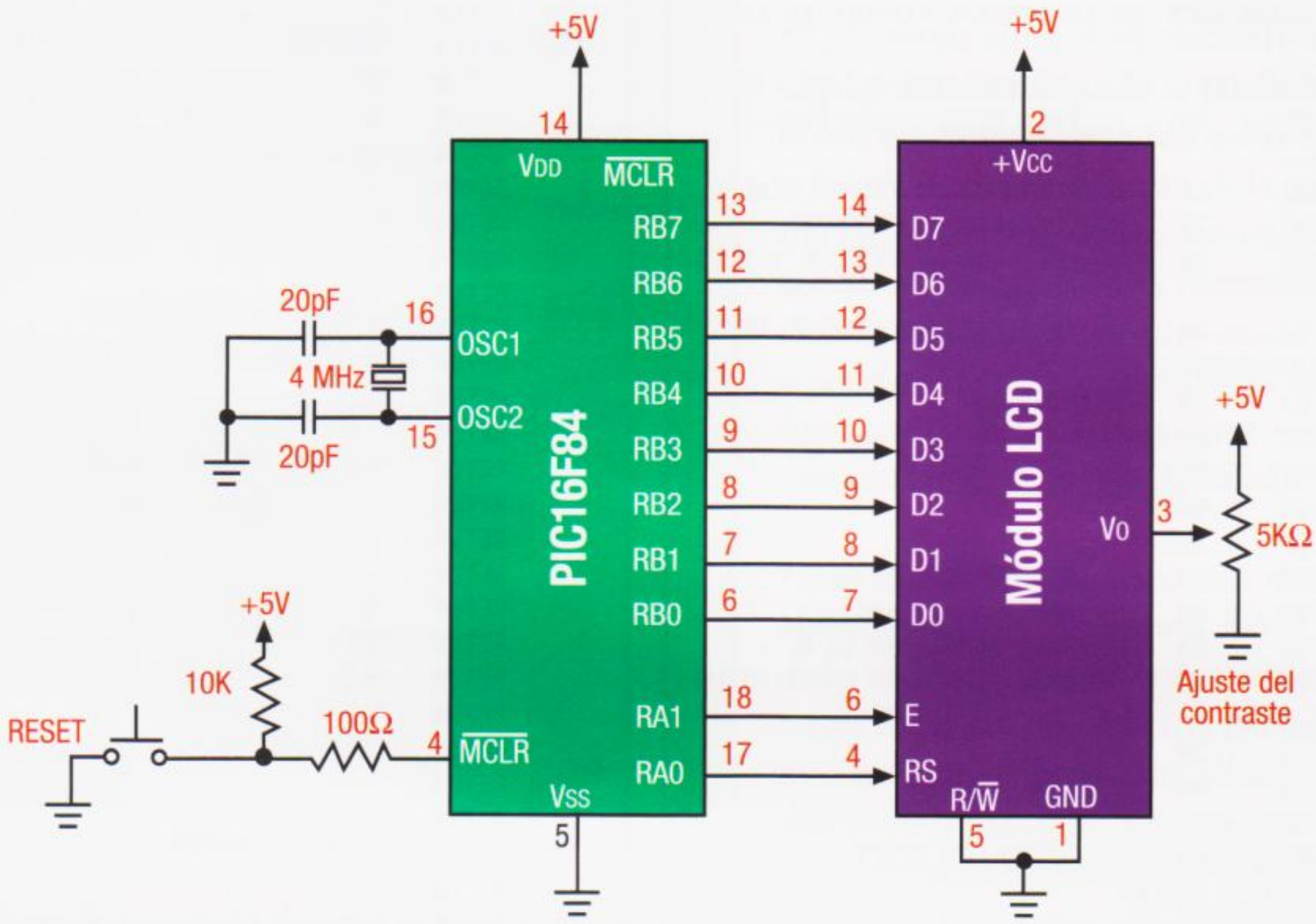


Figura 7.11 *Diagrama de conexiones entre el microcontrolador y el módulo LCD*

Para estos ejercicios en particular, sólo nos interesa escribir datos en la pantalla (no hacer lectura); por lo tanto, el pin de selección de lectura/escritura (R/W) en el *display*, se conecta a tierra. El puerto B del microcontrolador se utiliza como bus de datos, y el puerto A se encarga de generar las señales de control.

Para el oscilador del PIC16F84 se emplea un cristal de 4 MHz, por lo tanto tenemos ciclos de instrucción de un microsegundo. Para controlar el contraste de la pantalla en el módulo LCD, se emplea un potenciómetro de 5Kohm, conectado entre +5V y tierra.

En la **figura 7.12** se muestra el listado del programa. Para este caso, el ejemplo consiste en hacer circular un mensaje en la línea superior de la pantalla. La explicación de los pasos contenidos en él es la siguiente:

1. Se programan los puertos según las conexiones que se tienen en el circuito.
2. Se debe inicializar el módulo LCD. El primer dato que se envía (30H) le dice al módulo que la comunicación es a 8 *bits* y que se empleará solo una línea de caracteres. La rutina llamada CONTROL se encarga de generar las señales y los tiempos necesarios para que exista una correcta comunicación.
3. El segundo dato (07H) le dice al módulo que el mensaje se va a desplazar en la pantalla.
4. El tercer dato (0CH) hace que se encienda el *display*.
5. El siguiente paso es entrar en un ciclo que hace una lectura de la tabla donde se encuentra el mensaje y lo lleva a la memoria del módulo LCD. Cuando se terminan de enviar todos los caracteres, se inicia el ciclo nuevamente.

Las rutinas CONTROL y DATO emplean las mismas instrucciones. La única diferencia es que cada una le da el nivel lógico adecuado al pin RS, que indica si el dato enviado es un carácter del mensaje (un dato) o una instrucción de control.

En el modo que desplaza el mensaje en la pantalla se tiene un tiempo de espera un poco largo antes de que aparezcan los caracteres en ella, esto se debe a que primero se llena o se carga la memoria de datos de la parte no visible.

```
;ESTE PROGRAMA HACE QUE UN MENSAJE CIRCULE EN LA
PANTALLA
;DE UN MODULO LCD        ; P=16F84, OSC= XT, WDT = OFF
INDF    EQU     0H       ;PARA DIRECCIONAMIENTO
INDIRECTO
TMR0    EQU     1H       ;CONTADOR DE TIEMPO REAL
PC      EQU     2H       ;CONTADOR DE PROGRAMA
STATUS  EQU     3H       ;REGISTRO DE ESTADOS Y BITS
                         ;DE CONTROL
FSR     EQU     4H       ;SELECCCIÓN DE BANCOS DE
MEMORIA Y REGISTROS
PTOA    EQU     5H       ;PUERTOS
PTOB    EQU     6H
R0C     EQU     0CH      ;
R0D     EQU     0DH      ;
R13     EQU     13H      ;
Z       EQU     2H       ;BANDERA DE CERO
C       EQU     0H       ;BANDERA DE CARRY
W       EQU     0H       ;PARA ALMACENAR EN W
R       EQU     1H       ;PARA ALMACENAR EN EL MISMO
REGISTRO
E       EQU     1H
RS      EQU     0H

        ORG     00       ;VECTOR DE RESET
        GOTO    INICIO
        ORG     05H

RETARDO MOVLW   0FFH
        MOVWF   R13
DECRE   DECFSZ  R13,R
        GOTO    DECRE
        RETLW   0

CONTROL BCF     PTOA,RS  ;ESTA RUTINA GENERA LAS
                         ;SEÑALES DE CONTROL
        GOTO    DATO2    ;Y ENTREGA EL DATO
                         ;CORRESPONDIENTE AL MÓDULO
DATO    BSF     PTOA,RS  ;UTILIZA INTERFAZ A 8 BITS
DATO2   BSF     PTOA,E
        MOVWF   PTOB
        CALL    RETARDO
        BCF     PTOA,E
        CALL    RETARDO
        RETLW   0

TABLA2  ADDWF   PC,R     ;MENSAJE QUE DEBE SER ROTADO
        RETLW   "C"
        RETLW   "U"
        RETLW   "R"
        RETLW   "S"
        RETLW   "O"
        RETLW   " "
        RETLW   "D"
        RETLW   "E"
        RETLW   " "
        RETLW   "M"
        RETLW   "I"
        RETLW   "C"
        RETLW   "R"
        RETLW   "O"
        RETLW   "C"
        RETLW   "O"
        RETLW   "N"
        RETLW   "T"
        RETLW   "R"
        RETLW   "O"
        RETLW   "L"
        RETLW   "A"
        RETLW   "D"
        RETLW   "O"
        RETLW   "R"
        RETLW   "E"
        RETLW   "S"
        RETLW   " "
        RETLW   "P"
        RETLW   "I"
        RETLW   "C"
        RETLW   " "
        RETLW   "C"
        RETLW   "E"
        RETLW   "K"
        RETLW   "I"
        RETLW   "T"
        RETLW   " "
        RETLW   " "
        RETLW   " "
        RETLW   " "
        RETLW   " "
        RETLW   0

INICIO  MOVLW   0FCH     ;PROGRAMACIÓN DE PUERTOS
        TRIS    PTOA     ;SEGÚN EL CIRCUITO
        MOVLW   00H      ;
        TRIS    PTOB     ;
BEGIN   MOVLW   30H      ;INICIA DISPLAY A 8 BITS
                         ;Y 1 LÍNEA
        CALL    CONTROL
        MOVLW   07H      ;SELECCIONA EL MODO DE
                         ;DESPLAZAMIENTO
        CALL    CONTROL
        MOVLW   0CH      ;ACTIVA DISPLAY
        CALL    CONTROL
MUESTRA MOVLW   0        ;INICIA EL ENVIO DE
                         ;CARACTERES
        MOVWF   R0C      ;AL MÓDULO
CICLO   MOVF    R0C,W    ;HACE BARRIDO DE LA TABLA
        CALL    TABLA2
        CALL    DATO
        MOVLW   09FH     ;RETARDO ENTRE CARACTERES
        MOVWF   R0D
RETA1   CALL    RETARDO
        CALL    RETARDO
        DECFSZ  R0D,R
        GOTO    RETA1
        INCF    R0C,R    ;SIGUE CON EL PRÓXIMO
CARÁCTER DEL MENSAJE
        MOVLW   28H
        XORWF   R0C,W    ;PREGUNTA SI TERMINÓ EL
MENSAJE PARA VOLVER
        BTFSS   STATUS,Z ;A EMPEZAR
        GOTO    CICLO
        GOTO    MUESTRA
        END
```

Figura 7.12 *Programa para la conexión de 8 bits*

En varias lecciones de este curso hemos presentado poco a poco cada una de las instrucciones que componen el lenguaje de los microcontroladores PIC. Observemos ahora la sintaxis de las que se encuentran pendientes para completar todo el conjunto de las instrucciones y dar por finalizada esta lección.

ADDLW Suma un valor literal al contenido del registro W
Sintaxis: ADDLW k
Ejemplo: ADDLW 15
Antes de la instrucción: W=10
Después de la instrucción: W=25

ADDLW Suma el contenido de un registro al contenido del registro W
Sintaxis: ADDWF f,d
Ejemplo: ADDWF FSR,0
Antes de la instrucción: W=17 FSR = C2
Después de la instrucción: W=D9 FSR = C2

ANDLW Operación lógica AND entre un literal y el registro W
Sintaxis: ANDLW k
Ejemplo: ANDLW 5F
Antes de la instrucción: W=17 FSR = C2
Después de la instrucción: W=D9 FSR = C2

ANDWF Operación lógica AND entre un registro "f" y el registro W
Sintaxis: ANDWF f,d
Ejemplo: ANDWF FSR,1
Antes de la instrucción: W=17 FSR = C2
Después de la instrucción: W=17 FSR = 02

CLRW Borra el contenido del registro W, lo carga con 00
Sintaxis: CLRW
Ejemplo: CLRW
Antes de la instrucción: W=5A
Después de la instrucción: W=00

DECF Decrementa el contenido del registro "f"
Sintaxis: DECF f,d
Ejemplo: DECF regis, 1
Antes de la instrucción: regis = 13
Después de la instrucción: regis = 12

INCF Incrementa el contenido del registro "f"
Sintaxis: INCF f,d
Ejemplo: **INCF regis, 1**
Antes de la instrucción: regis = 24
Después de la instrucción: regis = 25

INCFSZ Incrementa el contenido del registro "f"; si el contenido de "f" queda en 00, salta una línea
Sintaxis: INCFSZ f,d
Ejemplo: etiqu1 INCFSZ regis, 1
goto etiqu2
....

Antes de la instrucción: contador de programa = etiqu1
Después de la instrucción: regis = regis + 1
Si regis = 0, entonces
contador de programa = etiqu1
Si regis <> 0, entonces
contador de programa = etiqu1 + 1

IORLW Operación lógica OR entre el registro W y el literal "k"
Sintaxis: IORLW k
Ejemplo: IORLW 35
Antes de la instrucción: W = 9A
Después de la instrucción: W = BF

IORWF Operación lógica OR entre el registro W y el registro "f"
Sintaxis: IORWF f,d
Ejemplo: IORWF regis,0
Antes de la instrucción: regis = 13 , W = 91
Después de la instrucción: regis = 13, W = 93

MOVF Mueve el contenido del registro "f"
Sintaxis: MOVF f,d
Ejemplo: MOVF regis,0
Antes de la instrucción: W=?
Después de la instrucción: W= valor guardado en regis

OPTION Carga el registro OPCION con el contenido del registro W
Sintaxis: OPTION
Ejemplo: OPTION
Esta instrucción se considera obsoleta, pero por facilidad en el aprendizaje, se utiliza en el nivel básico.

RETFIE Retorno del llamado a interrupción.
Sintaxis: RETFIE
Ejemplo: RETFIE
Antes de la instrucción:
Contador de programa = ?
Después de la instrucción:
Contador de programa = pila

SUBLW resta el contenido del registro W del literal "k"
Sintaxis: SUBLW k
Ejemplo 1: SUBLW 02
Antes de la instrucción: W=1, C=?
Después de la instrucción: W=1, C =1 (positivo)
Ejemplo 2:
Antes de la instrucción: W=2, C=?
Después de la instrucción: W=0, C =1 (cero)
Ejemplo 3:
Antes de la instrucción: W=3, C=?
Después de la instrucción: W=FF, C =0 (negativo)

SUBWF resta el contenido del registro W del registro "f"
Sintaxis: SUBWF f,d
Ejemplo 1: SUBLW regis,1
Antes de la instrucción: regis=3,W = 2, C=?
Después de la instrucción: regis=1, W = 2, C=1 (positivo)
Ejemplo 2:
Antes de la instrucción: regis=2, W = 2, C=?
Después de la instrucción: regis = 0, W = 2, C=1 (cero)
Ejemplo 3:
Antes de la instrucción: regis=1, W = 2, C=?
Después de la instrucción: regis =FF, W = 2, C=0 (negativo)

SWAPF Intercambia los cuatro *bits* altos y los cuatro *bits* bajos del registro "f"
Sintaxis: SWAPF f,d
Ejemplo: **SWAPF regis,0**
Antes de la instrucción: regis = A5 , W = ¿
Después de la instrucción: regis = A5, W =5A

TRIS Carga el registro TRIS (programación de los puertos como entrada/salida) con el contenido del registro W
Sintaxis: TRIS f
Ejemplo: TRIS puertoA
Esta instrucción se considera obsoleta, pero por facilidad en el aprendizaje, se utiliza en el nivel básico.

XORLW Operación lógica XOR entre el registro W y el literal "k"
Sintaxis: XORLW k
Ejemplo: XORLW AF
Antes de la instrucción: W = B5
Después de la instrucción: W= 1A

XORWF Operación lógica XOR entre el registro W y el registro "f"
Sintaxis: XORWF f,d
Ejemplo: XORWF regis,1
Antes de la instrucción: regis = AF, W = B5
Después de la instrucción: regis = 1A, W =B5

LECCIÓN 8

EJERCICIOS CON MICROCONTROLADORES MOTOROLA

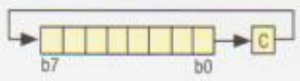
A partir de esta lección iniciaremos una serie de ejercicios con los microcontroladores Motorola. Debido a su arquitectura CISC, los microcontroladores de la familia HC08 cuentan con una gran cantidad de instrucciones. Para poder desarrollar mejor los ejercicios, podemos tenerlas como referencia en la tabla con que iniciamos esta lección.

Forma	Operación	Descripción	Efecto sobre CCR						Ciclos
			V	H	I	N	Z	C	
ADC #opr	Sumar con *carry*	A←(A) + (M) + (C)	↕	↕	-	↕	↕	↕	2
ADC opr									3
ADC opr									4
ADC opr,X									4
ADC opr,X									3
ADC ,X									2
ADC opr,SP									4
ADC opr,SP									5
ADD #opr	Sumar con *carry*	A←(A) + (M)	↕	↕	-	↕	↕	↕	2
ADD opr									3
ADD opr									4
ADD opr,X									4
ADD opr,X									3
ADD ,X									2
ADD opr,SP									4
ADD opr,SP									5
AIS #opr	Sumar el valor al SP	SP←(SP) + (16 « M)	-	-	-	-	-	-	2
AIX #opr	Sumar el valor a H:X	H:X←(H:X) + (16 « M)	-	-	-	-	-	-	2
AND #opr	AND lógica	A←(A) & (M)	0	-	-	↕	↕	-	2
AND opr									3
AND opr									4
AND opr,X									4
AND opr,X									3
AND ,X									2
AND opr,SP									4
AND opr,SP									5
ASL opr	Rotación a la izquierda	C ← b7 … b0 ←	↕	-	-	↕	↕	↕	4
ASLA									1
ASLX									1
ASL opr,X									4
ASL ,X									3
ASL opr,SP									5
ASR opr	Rotación a la derecha	→ b7 … b0 → C	↕	-	-	↕	↕	↕	4
ASRA									1
ASRX									1
ASR opr,X									4
ASR opr,X									3
ASL opr,SP									5
BCC rel	Saltar si *carry* es cero	PC← (PC)+$0002+rel?(C)=0	-	-	-	-	-	-	3
BCLR n, opr	Borrar *bit* en memoria	Mn← 0	-	-	-	-	-	-	4
BCS rel	Saltar si *carry* es uno	PC← (PC)+$0002+rel?(C)=1	-	-	-	-	-	-	3
BEQ rel	Saltar si es igual	PC← (PC)+$0002+rel?(Z)=1	-	-	-	-	-	-	3
BGE opr	Saltar si es mayor o igual a	PC← (PC)+$0002+rel?(N ⊕ V)=0	-	-	-	-	-	-	3
BGT opr	Saltar si es mayor a	PC← (PC)+$0002+rel?(Z) I (N ⊕ V)=0	-	-	-	-	-	-	3
BHCC rel	Saltar si el bit de half carry está en cero	PC← (PC)+$0002+rel?(H)=0	-	-	-	-	-	-	3
BHCS rel	Saltar si bit half está en uno	PC← (PC)+$0002+rel?(H)=1	-	-	-	-	-	-	3

Forma	Operación	Descripción	Efecto sobre CCR						Ciclos
			V	H	I	N	Z	C	
BHI rel	Saltar si es el mayor	PC ← (PC)+$0002+rel?(C)|(Z)=0	-	-	-	-	-	-	3
BHS rel	Saltar si es mayor o igual	PC ← (PC)+$0002+rel?(C)=0	-	-	-	-	-	-	3
BIH rel	Saltar si el pin $\overline{IRQ}$ está en alto	PC ← (PC)+$0002+rel?$\overline{IRQ}$=1	-	-	-	-	-	-	3
BIL rel	Saltar si el pin $\overline{IRQ}$ está en bajo	PC ← (PC)+$0002+rel?$\overline{IRQ}$=0	-	-	-	-	-	-	3
BIT #opr	Probar bit	(A) & (M)	0	-	-	↕	↕	-	2
BIT opr									3
BIT opr									4
BIT opr,X									4
BIT opr,X									3
BIT ,X									2
BIT opr,SP									4
BIT opr,SP									5
BLE opr	Saltar si es menor ó igual	PC ← (PC)+$0002+rel?(Z) | (N ⊕ V)=1	-	-	-	-	-	-	3
BLO rel	Saltar si es el más bajo	PC ← (PC)+$0002+rel?(C)=1	-	-	-	-	-	-	3
BLS rel	Saltar si es menor ó igual	PC ← (PC)+$0002+rel?(C)|(Z)=1	-	-	-	-	-	-	3
BLT opr	Saltar si es menor que	PC ← (PC)+$0002+rel?(N ⊕ V)=1	-	-	-	-	-	-	3
BMC rel	Saltar si la máscara de la interrupción está en cero	PC ← (PC)+$0002+rel?(I)=0	-	-	-	-	-	-	3
BMI rel	Saltar si menos	PC ← (PC)+$0002+rel?(N)=1	-	-	-	-	-	-	3
BMS rel	Saltar si la máscara de la interrupción está activada	PC ← (PC)+$0002+rel?(I)=1	-	-	-	-	-	-	3
BNE rel	Saltar si no es igual	PC ← (PC)+$0002+rel?(Z)=0	-	-	-	-	-	-	3
BPL rel	Saltar si más	PC ← (PC)+$0002+rel?(N)=0	-	-	-	-	-	-	3
BRA rel	Saltar siempre	PC ← (PC)+$0002+rel	-	-	-	-	-	-	35
BRCLR n,opr,rel	Saltar si el bit n en memoria está en cero	PC ← (PC)+$0003+rel?(Mn)=0	-	-	-	-	-	↕	3
BRN rel	Nunca saltar	PC ← (PC)+$0002	-	-	-	-	-	-	5
BRSET n,opr,rel	Saltar si el bit n en memoria está en uno	PC ← (PC)+$0003+rel?(Mn)=1	-	-	-	-	-	↕	4
BSET n,opr	Activar bit n en la memoria	Mn ← 1	-	-	-	-	-	-	4
BSR rel	Saltar a subrutina	PC ← (PC)+$0002; empuje(PCL) SP ← (SP)-$0001; empuje(PCH) SP ← (SP)-$0001 PC ← (PC)+rel	-	-	-	-	-	-	4
CBEQ opr,rel	Compare y salte y es igual	PC ← (PC)+$0003+rel?(A)-(M)=$00	-	-	-	-	-	-	5
CBEQA #opr,rel		PC ← (PC)+$0003+rel?(A)-(M)=$00							4
CBEQX #opr,rel		PC ← (PC)+$0003+rel?(X)-(M)=$00							4
CBEQ opr;X^+,rel		PC ← (PC)+$0003+rel?(X)-(M)=$00							5
CBEQ X^+,rel		PC ← (PC)+$0002+rel?(X)-(M)=$00							4
CBEQ opr;SP,rel		PC ← (PC)+$0004+rel?(X)-(M)=$00							6
CLC	Borrar *bit carry*	C ← 0	-	-	-	-	-	0	1
CLI	Borrar *bit* de la máscara de la interrupción	I ← 0	-	-	0	-	-	-	2
CLR opr	Borrar	M ← $00	0	-	-	0	1	-	3
CLRA		A ← $00							1
CLRX		X ← $00							1
CLRH		H ← $00							1
CLR opr,X		M ← $00							3
CLR ,X		M ← $00							2
CLR opr,SP		M ← $00							4
CMP #opr	Compara acumulador con memoria	(A)-(M)	↕	-	-	↕	↕	↕	2
CMP opr									3
CMP opr									4
CMP opr,X									4
CMP opr,X									3
CMP ,X									2
CMP opr,SP									4
CMP opr,SP									5

Forma	Operación	Descripción	Efecto sobre CCR						Ciclos
			V	H	I	N	Z	C	
COM opr	Complementar (complemento a	M ← ($\overline{M}$)=$FF-(M)	0	-	-	↕	↕	1	4
COMA	uno)	A ← ($\overline{A}$)=$FF-(M)							1
COMX		X ← ($\overline{X}$)=$FF-(M)							1
COM opr,X		M ← ($\overline{M}$)=$FF-(M)							4
COM ,X		M ← ($\overline{M}$)=$FF-(M)							3
COM opr,SP		M (M)=$FF-(M)							5
CPHX #opr	Compare el registro Index (H:X)	(H:X)-(M:M+$0001)	↕	-	-	↕	↕	↕	3
CPHX opr	con la memoria								4
CPX #opr	Compare X(Registro index bajo)	(X)-(M)	↕	-	-	↕	↕	↕	2
CPX opr	con memoria								3
CPX opr									4
CPX ,X									4
CPX opr,X									3
CPX opr,X									2
CPX opr,SP									4
CPX opr,SP									5
DAA	Ajuste decimal del acumulador	$(A)_{10}$	U	-	-	↕	↕	↕	2
DBNZ opr,rel	Decremente y salte si no es cero	A ← (A)-$0001 ó M ← (M)-$01 ó X ← (X)-$0001	-	-	-	-	-	-	5
DBNA rel		PC ← (PC)+$0003+rel si(Resultado)≠0							3
DBNZX rel		para DBNZ directo, IX1							3
DBNZ opr,X,rel		PC ← (PC)+$0002+rel si(Resultado)≠0							5
DBNZ X,rel		para DBNZA,DBNZX, ó IX							4
DBNZ opr,SP,rel		PC ← (PC)+$0004+rel si(Resultado)≠0							6
		para DBNZ SP1							
DEC opr	Decremente	M ← (M)-$01	↕	-	-	↕	↕	-	4
DECA		A ← (A)-$01							1
DECX		X ← (X)-$01							1
DEC opr,X		M ← (M)-$01							4
DEC ,X		M ← (M)-$01							3
DEC opr,SP		M ← (M)-$01							5
DIV	Divide	A ← (H:A)/(X)	-	-	-	-	↕	↕	7
		H ← Remanente							
EOR #opr	OR exclusivo	A ← (A ⊕ M)	0	-	-	↕	↕	-	2
EOR opr	memoria con acumulador								3
EOR opr									4
EOR opr,X									4
EOR opr,X									3
EOR ,X									2
EOR opr,SP									4
EOR opr,SP									5
INC opr	Incremente	M ← (M)+$01	↕	-	-	↕	↕	-	4
INCA		A ← (A)+$01							1
INCX		X ← (X)+$01							1
INC opr,X		M ← (M)+$01							4
INC ,X		M ← (M)+$01							3
INC opr,SP		M ← (M)+$01							5
JMP opr	Salte	PC ← Salte dirección	-	-	-	-	-	-	2
JMP opr									3
JMP opr,X									4
JMP opr,X									3
JMP ,X									2
JSR opr	Salte a subrutina	PC ← (PC)+n(n=1,2 ó 3)	-	-	-	-	-	-	4
JSR opr		Pone (PCL); SP ← (SP)-$0001							5
JSR opr,X		Pone (PCH); SP ← (SP)-$0001							6
JSR opr,X		PC ← dirección incondicional							5
JSR ,X									4

Forma	Operación	Descripción	Efecto sobre CCR						Ciclos
			V	H	I	N	Z	C	
LDA #opr	Carga acumulador desde la	A ← (M)	0	-	-	↕	↕	-	2
LDA opr	memoria								3
LDA opr									4
LDA opr,X									4
LDA opr,X									3
LDA ,X									2
LDA opr,SP									4
LDA opr,SP									5
LDHX #opr	Carga registro índice (H:X) desde	H:X ← (M:M+$0001)	0	-	-	↕	↕	-	3
LDHX opr	la memoria								4
LDX #opr	Carga X (registro índice bajo)	X ← (M)	0	-	-	↕	↕	-	2
LDX opr	desde la memoria								3
LDX opr									4
LDX opr,X									4
LDX opr,X									3
LDX ,X									2
LDX opr,SP									4
LDX opr,SP									5
LSL opr	Desplazamiento lógico a la	C ← b7 … b0 ← 0	↕	-	-	↕	↕	↕	4
LSLA	izquierda (igual que ASL)								1
LSLX									1
LSL opr,X									4
LSL ,X									3
LSL opr,SP									5
LSR opr	Desplazamiento lógico a la	0 → b7 … b0 → C	↕	-	-	0	↕	↕	4
LSRA	derecha								1
LSRX									1
LSR opr,X									4
LSR ,X									3
LSR opr,SP									5
MOV opr,opr	Mover	(M)destino ← (M)origen	0	-	-	↕	↕	-	5
MOV opr,X+		H:X ← (H:X)+$001 en IX+D y DIX+Modos							4
MOV #opr,opr									4
MOV X+,opr									4
MUL	Multiplicación sin signo	X:A ← (X) x (A)	-	0	-	-	-	0	5
NEG opr	Negar	M ← -(M)=$00-(M)	↕	-	-	↕	↕	↕	4
NEGA		A ← -(A)=$00-(A)							1
NEGX		X ← -(X)=$00-(X)							1
NEG opr,X		M ← -(M)=$00-(M)							4
NEG ,X		M ← -(M)=$00-(M)							3
NEG opr,SP		M ← -(M)=$00-(M)							5
NOP	No operación	Ninguno	-	-	-	-	-	-	1
NSA	Intercambio de nibbles en el acumulador	A ← (A[3:0]:A[7:4])	-	-	-	-	-	-	3
ORA #opr	OR entre el acumulador y la	A ← (A)|(M)	0	-	-	↕	↕	-	2
ORA opr	memoria								3
ORA opr									4
ORA opr,X									4
ORA opr,X									3
ORA ,X									2
ORA opr,SP									4
ORA opr,SP									5
PSHA	Pone el acumulador dentro de la pila	Poner (*push*) (A); SP ← (SP)-$0001	-	-	-	-	-	-	2
PSHH	Pone H (registro índice alto) en la pila	Poner (H); SP ← (SP)-$0001	-	-	-	-	-	-	2
PSHX	Pone X (registro índice bajo) en la pila	Poner (X); SP ← (SP)-$0001	-	-	-	-	-	-	2

Forma	Operación	Descripción	Efecto sobre CCR						Ciclos
			V	H	I	N	Z	C	
PULA	Acumulador toma desde la pila	SP ← (SP+$0001); atrae(A)	-	-	-	-	-	-	2
PULH	Toma H(registro index bajo) desde la pila	SP ← (SP+$0001); atrae(H)	-	-	-	-	-	-	2
PULX	Toma X(registro index bajo) desde la pila	SP ← (SP+$0001); atrae(X)	-	-	-	-	-	-	2
ROL opr	Rotar a la izquierda a través de carry		↕	-	-	↕	↕	↕	4
ROLA									1
ROLX									1
ROL opr,X									4
ROL ,X									3
ROL opr,SP									5
ROR opr	Rotar a la derecha a través de carry		↕	-	-	↕	↕	↕	4
RORA									1
RORX									1
ROR opr,X									4
ROR ,X									3
ROR opr,SP									5
RSP	*Reset* SP	SP ← $FF	-	-	-	-	-	-	1
RTI	Retorne desde la interrupción	SP ← (SP)+$0001; Atrae(CCR) SP ← (SP)+$0001; Atrae(A) SP ← (SP)+$0001; Atrae(X) SP ← (SP)+$0001; Atrae(PCH) SP ← (SP)+$0001; Atrae(PCL)	↕	↕	↕	↕	↕	↕	7
RTS	Retorne desde la subrutina	SP ← (SP)+$0001; Atrae(PCH) SP ← (SP)+$0001; Atrae(PCL)	-	-	-	-	-	-	4
SBC #opr	Restar con *carry*	A ← (A)-(M)-(C)	↕	-	-	↕	↕	↕	2
SBC opr									3
SBC opr									4
SBC opr,X									4
SBC opr,X									3
SBC ,X									2
SBC opr,SP									4
SBC opr,SP									5
SEC	Bit *carry* activado	C ← 1	-	-	-	-	-	1	1
SEI	Activar máscara de la interrupción	I ← 1	-	-	1	-	-	-	2
STA opr	Guardar acumulador en memoria	M ← (A)	0	-	-	↕	↕	-	3
STA opr									4
STA opr,X									4
STA opr,X									3
STA ,X									2
STA opr,SP									4
STA opr,SP									5
STHX opr	Guarda H:X (registro índice)	(M:M+$0001) ← (H:X)	0	-	-	↕	↕	-	4
STOP	Deshabilita pin $\overline{IRQ}$; Detiene Osc.	I bit ← 0; Detiene oscilador	-	-	0	-	-	-	1
STX opr	Guarda X (registro índice bajo) en memoria	M ← (X)	0	-	-	↕	↕	-	3
STX opr									4
STX opr,X									4
STX opr,X									3
STX ,X									2
STX opr,SP									4
STX opr,SP									5
SUB #opr	Restar	A ← (A)-(M)	↕	-	-	↕	↕	↕	2
SUB opr									3
SUB opr									4

Forma	Operación	Descripción	Efecto sobre CCR						Ciclos
			V	H	I	N	Z	C	
SUB opr,X		SP ← (SP+$0001); atrae(A)							3
SUB opr,X									2
SUB ,X									4
SUB opr,SP									5
SUB opr,SP									4
SWI	Interrupción por programa	PC ← (PC)+$0001; Pone(PCL) SP ← (SP)-$0001; Pone(PCH) SP ← (SP)-$0001; Pone(X) SP ← (SP)-$0001; Pone(A) SP ← (SP)-$0001; Pone(CCR) SP ← (SP)-$0001; I ← 1 SP ← Vector de interrupción para byte en alto SP ← Vector de interrupción para byte en bajo	-	-	1	-	-	-	9
TAP	Transferir acumulador a CCR	CCR ← (A)	↕	↕	↕	↕	↕	↕	2
TAX	Transferir acumulador a X (registro índice bajo)	X ← (A)	-	-	-	-	-	-	1
TPA	Transferir CCR al acumulador	A ← (CCR)	-	-	-	-	-	-	1
TST opr	Prueba para negación ó cero	(A) ← -$00	0	-	-	↕	↕	-	3
TSTA		(X) ← -$00							1
TSTX		(M) ← -$00							1
TST opr,X									3
TST ,X									2
TST opr,SP									4
TSX	Transferir SP a registro índice	H:X ← (SP)+$0001	-	-	-	-	-	-	2
TXA	Transferir X (registro índice bajo) hacia el acumulador	A ← (X)	-	-	-	-	-	-	1
TXS	Transferir registro índice a SP	(SP) ← (H:X)-$0001	-	-	-	-	-	-	2
WAIT	Deshabilitar interrupciones; parar procesador	1 *bit* ← 0	-	-	0	-	-	-	1

Tabla 8.1 *Continuación del conjunto de instrucciones de la familia HC08*

Nomenclatura

A continuación se listan las nomenclaturas utilizadas dentro del anterior conjunto de instrucciones.

Operadores

() = Contenidos del registro o posición de memoria ubicada dentro del paréntesis
← = Es cargado con
& = Operación AND
| = Operación OR
⊕ = OR exclusiva
x = Multiplicación
÷ = División
: = Concatenar
+ = Sumar
- = Negar (complemento a dos)
« = Signo extendido
A = Acumulador
CCR = Registro de condiciones
H = Parte alta del registro índice
X = Parte baja del registro índice
PC = Contador de programa
SP = Puntero a la pila (*stack pointer*)
OPR = Operando

Para iniciar el estudio de la programación de los microcontroladores Motorola, vamos a desarrollar una secuencia de ejercicios con los cuales aprenderemos paso a paso el manejo de las principales funciones que se pueden llevar a cabo mediante este tipo de circuitos integrados.

Ejercicio No. 9
Energizar un diodo LED

El objetivo de este ejercicio es energizar un diodo LED conectado a una línea del puerto B del microcontrolador Motorola 68HC908JK3. A pesar de su simplicidad, este proyecto contiene ideas importantes que luego van a ser clave en el desarrollo de proyectos de mayor complejidad.

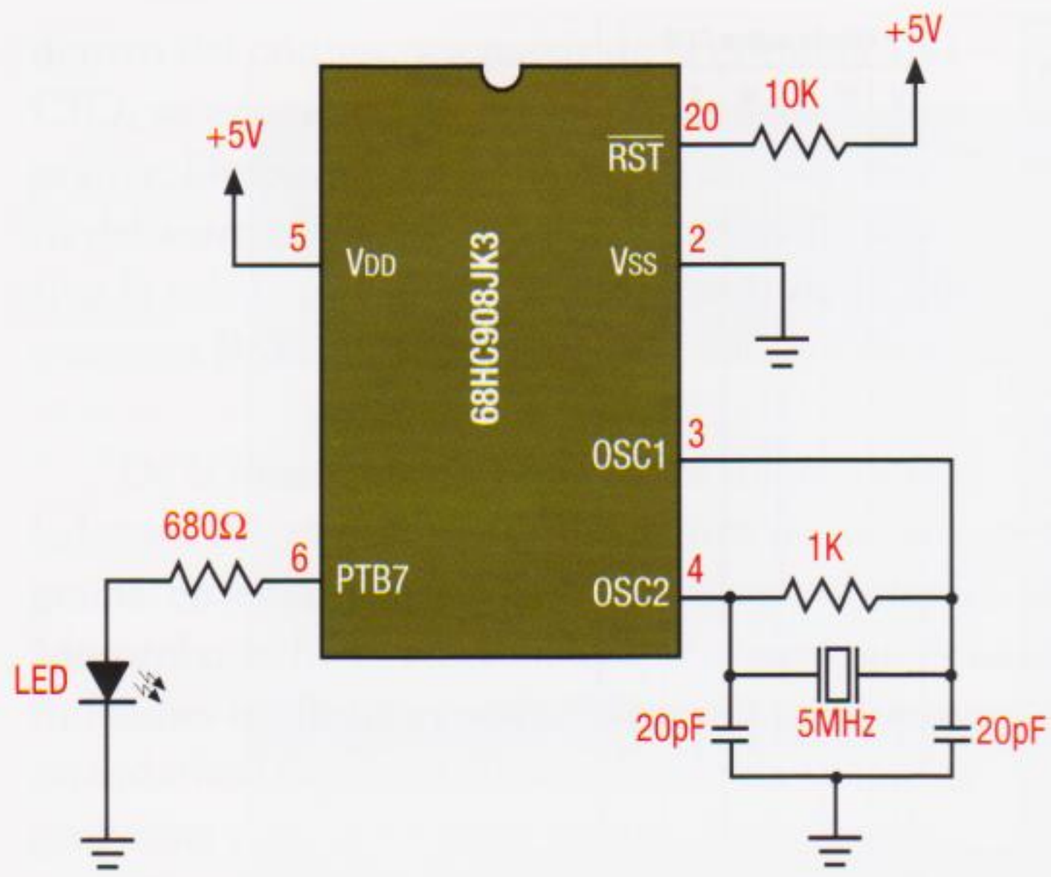


Figura 8.1 *Circuito para energizar un diodo LED, empleando un microcontrolador Motorola*

Teniendo en cuenta la arquitectura del microcontrolador 68HC908JK3 se puede deducir cual puede ser la estructura física del circuito que resuelve el problema de energizar un diodo LED. Éste es el paso inicial en el desarrollo del proyecto. La **figura 8.1** muestra el diseño físico el cual se puede montar en un *protoboard*.

El sistema de desarrollo Motorola K-218, que estudiamos en el proyecto 8 de este curso, permite ejecutar una simulación dentro del circuito por medio de un programa de computadora, **figura 8.2**.

Como se ha mencionado en anteriores oportunidades, este programa cuenta con una línea de comandos a través de la cual podemos, entre otras funciones, configurar el funcionamiento de los puertos y originar señales de salida, o leer datos de entrada. Por ejemplo, para configurar el puerto D podemos utilizar la instrucción DDRD. Conectemos la tarjeta al puerto serial de una computadora y sigamos el procedimiento mostrado en el proyecto 8 para poner la tarjeta en línea. Sigamos el procedimiento que se menciona a continuación.

1. Por medio del ratón ubicamos el cursor en la línea de comandos, localizada en la esquina inferior izquierda.
2. En esta línea configuremos el puerto D, pulsando DDRD 80. Con esto se configura el pin 7 del puerto D como salida 7 y los demás pines como entradas, **figura 8.3**.
3. Ahora escribamos el comando PORTD 80 para activar el pin de salida 7 del puerto D. En este momento el diodo LED indicador del circuito se debe encender, **figura 8.4**.
4. Ahora escribamos el comando PORTD 00 para desactivar la misma línea y apagar el LED, **figura 8.5**.

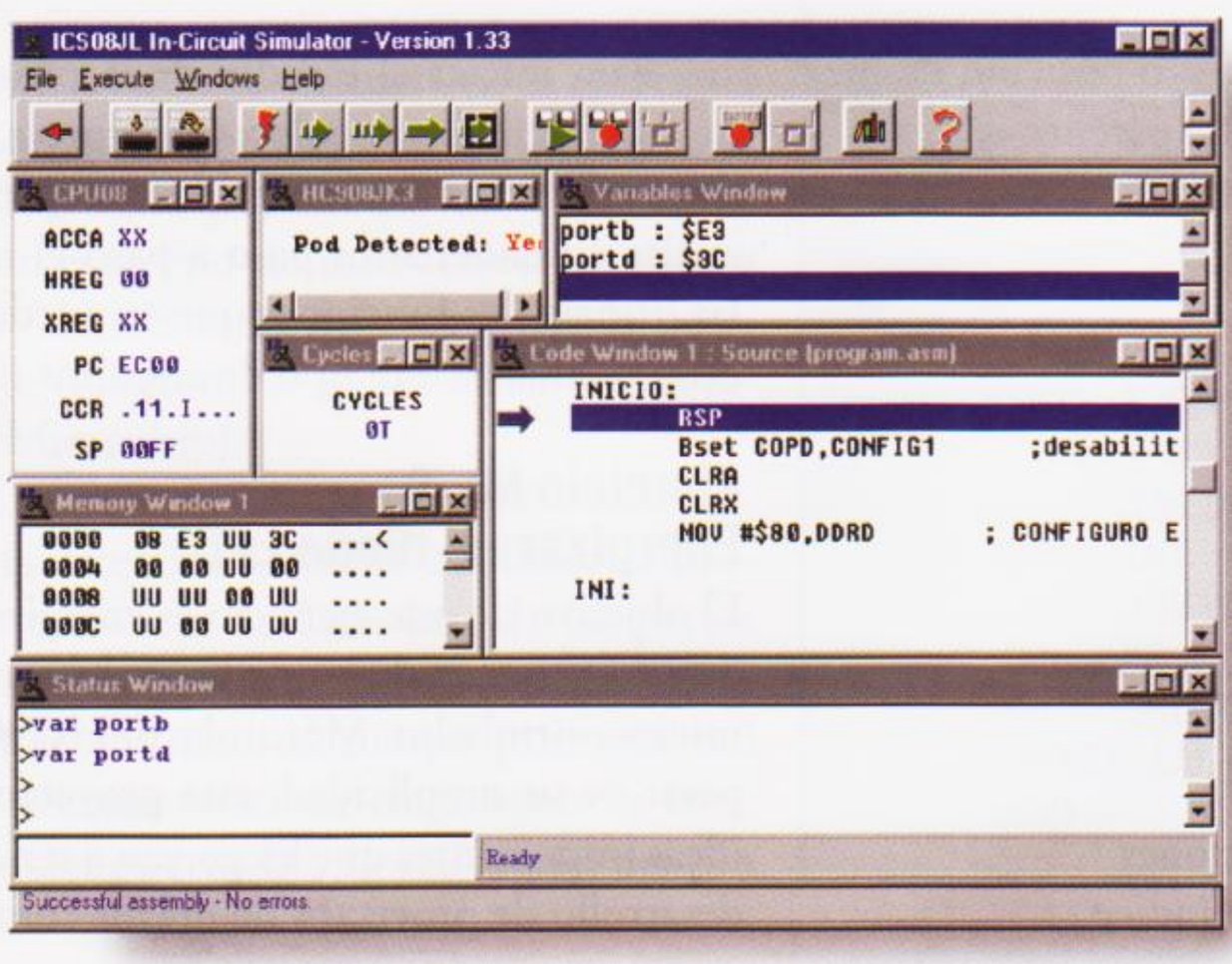


Figura 8.2 *Programa WinIDE de la empresa PEMICRO*

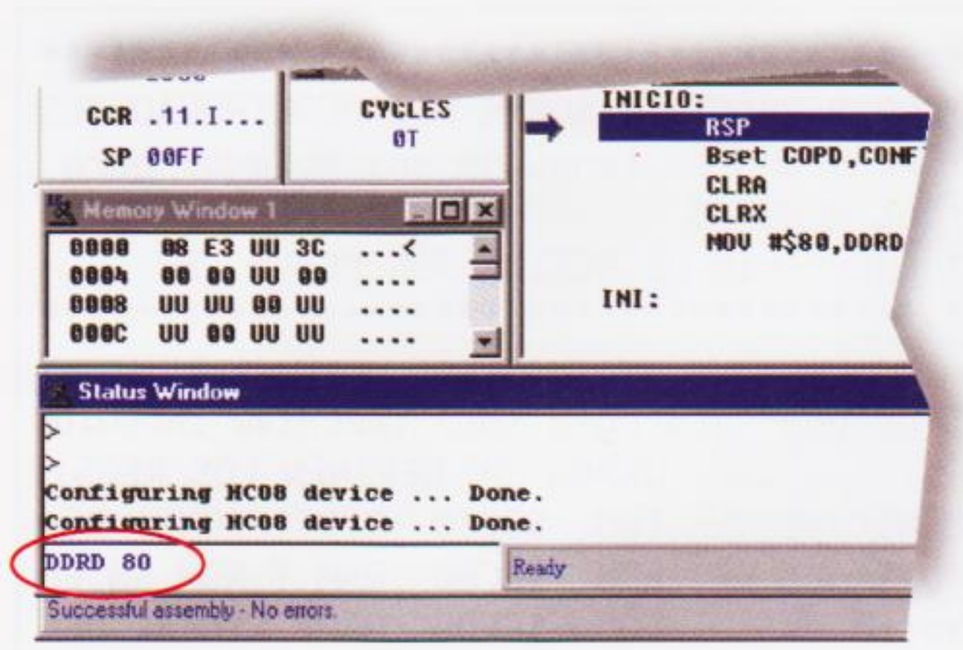


Figura 8.3 Configuración del puerto D

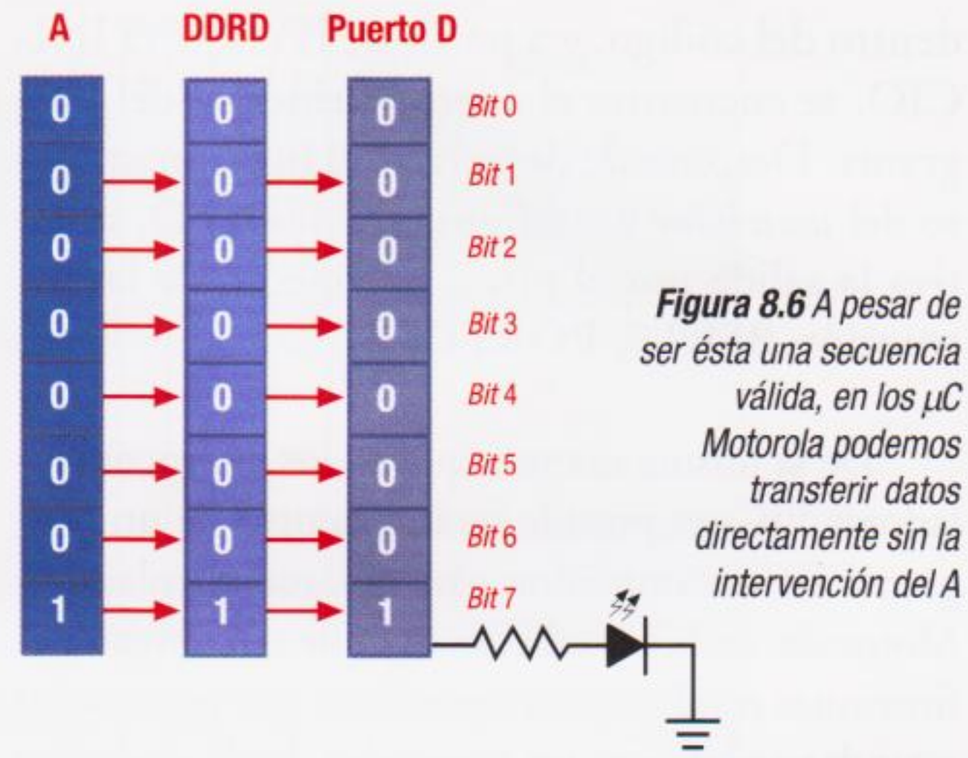


Figura 8.6 A pesar de ser ésta una secuencia válida, en los µC Motorola podemos transferir datos directamente sin la intervención del A

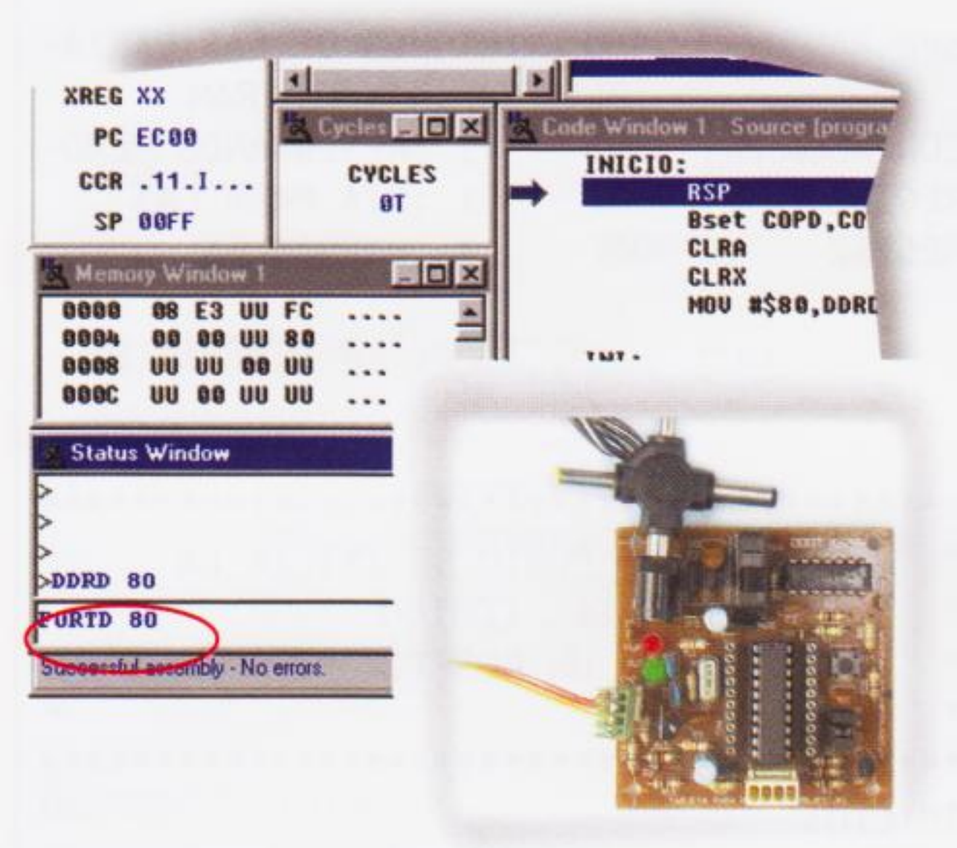


Figura 8.4 Activando el pin 7 del puerto D

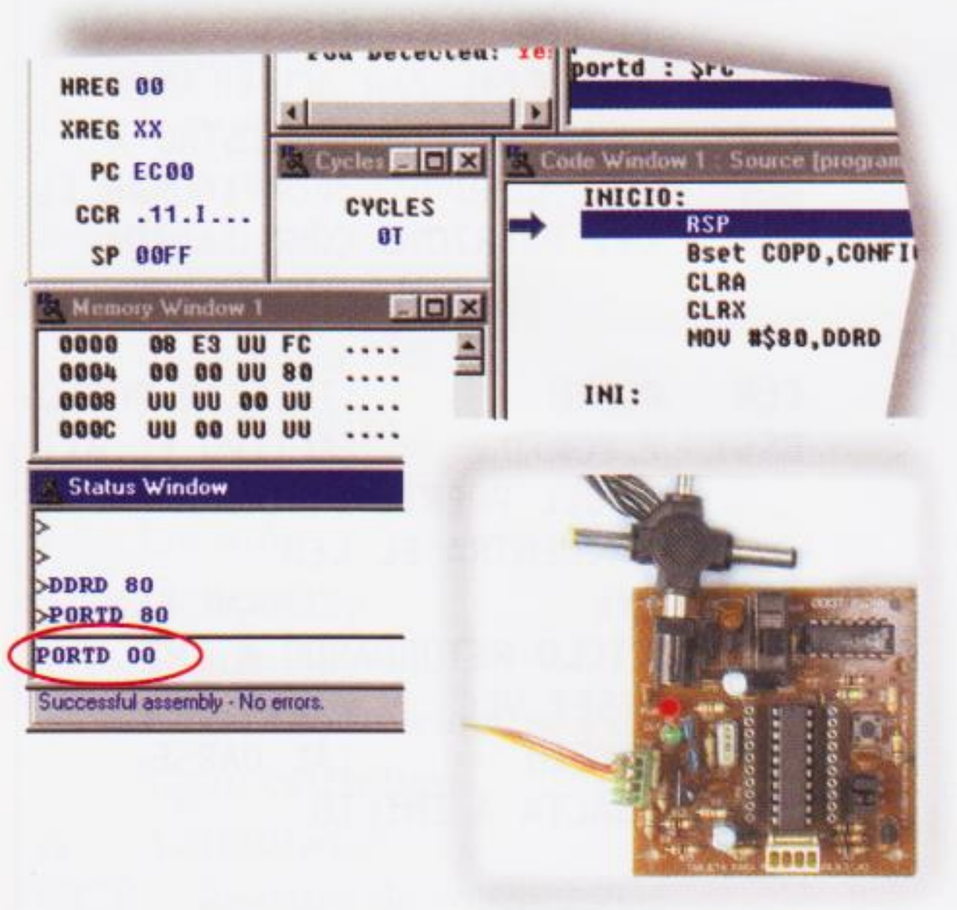


Figura 8.5 Desactivando el pin 7 del puerto D

El propósito del proyecto es, en pocas palabras, transferir el dato binario 10000000 al puerto D. Recordemos que para transferir información entre el microcontrolador y su entorno se hace a través de registros. Por esta razón es comprensible que se utilizara la siguiente secuencia:

0x80 → Registro de trabajo
Registro de trabajo → puerto D

En la **figura 8.6** se muestra un diagrama que refuerza este concepto. Los microcontroladores Motorola, dentro de sus modos de direccionamiento, permiten transferir información directamente entre dos posiciones de memoria, evitándonos esta secuencia.

Contrario a lo que ocurre en los microcontroladores PIC, la arquitectura Motorola posee un mapa lineal de memoria, por lo cual no se presenta la necesidad de direccionar bancos en memoria RAM, ni páginas en memoria ROM.

En la **figura 8.7** apreciamos el código fuente de este ejercicio. En él podemos establecer cierta similitud existente con respecto a un programa escrito en lenguaje ensamblador para un microcontrolador de Microchip.

El programa empieza estableciendo las direcciones dentro del mapa lineal de memoria, de cada una de las porciones del programa; luego se definen los registros que se van a utilizar

dentro del código, y a partir de la etiqueta INICIO, se encuentra el cuerpo principal del programa. Después de desactivar el funcionamiento del *watchdog* y configurar el puerto D, se activa la salida por el pin 7 por medio de la instrucción **BSET7, PORTD**.

De la misma manera que en los microcontroladores PIC, es posible incluir dentro de un programa en ensamblador para microcontroladores Motorola, archivos adicionales que contengan definiciones o subrutinas específicas que puedan ser estandarizadas para ser invocadas desde cualquier programa y así ahorrarnos tiempo de desarrollo.

En la primera línea de este código podemos apreciar la inclusión del archivo **jl3regs.inc**, el cual puede descargarse libremente de nuestra página en la internet www.cekit.com.co, en el cual se define el mapa de memoria y los registros específicos de este microcontrolador. Esto lo podemos hacer gracias a la directiva **include**.

Dentro del programa es importante definir las direcciones de cada uno de los espacios de memoria RAM y de memoria ROM, por ello se definen estos valores por medio de la directiva **EQU**, de acuerdo a lo expuesto en el mapa lineal de memoria de esta familia de microcontroladores.

A continuación se definen en la memoria RAM los registros que van a ser utilizados dentro del programa, por medio de la directiva **RMB**. Con este tipo de instrucción se reserva un espacio de memoria sin especificar la posición exacta (tal como se debía hacer en las familias de Microchip).

Con la directiva **ORG** se establece la ubicación en memoria de cada una de las partes del programa, a partir de la instrucción **ORG RomStart** se da inicio al cuerpo principal del mismo. Podemos observar claramente la configuración del puerto D y la activación del pin 7 dentro de la etiqueta **INI**.

```
*************************************
*CURSO PRÁCTICO SOBRE MICROCONTROLADORES
* Programa en lenguaje ensamblador para
* el HC08JK3
* EJERCICIO DE ACTIVACIÓN DE UN LED
* *************************************

$include 'jl3regsg.inc' ;ARCHIVO INCLUIDO
             ;DONDE SE DEFINEN LOS REGS.
RAMStart     EQU  $0080 ;DIRECCIONES DE
RomStart     EQU  $EC00 ;RAM Y ROM EN EL
Vectores     EQU  $FFDE ;MAPA LINEAL DE
;RomFin      EQU  $FB00 ;MEMORIA
RESET_VEC    EQU  $FFFE

org RAMStart ;DEFINICIONES DE LAS VARIA-
              ;BLES EN MEMORIA RAM
CONTADOR     RMB     1 ;RESERVANDO MEMO-
REGIS1       RMB     1 ;RIA PARA LAS
REGIS2       RMB     1 ;VARIABLES

org RomStart ; INICIO DE LA MEMORIA DE
              ; PROGRAMA

*************************************
*  - DESDE ESTE PUNTO SE INICIA LA
*    EJECUCIÓN DEL CÓDIGO            *
*    DESPUÉS DE UN RESET.            *
*                                    *
*************************************
INICIO:
       RSP   ; INICIALIZAR EL PUNTERO A
             ;LA PILA (SP)
       Bset  COPD,CONFIG1  ;DESHABILITA
             ;EL COP (WATCHDOG)
       CLRA  ; INICIALIZAR ACUMULADOR
       CLRX  ; INICIALIZAR REGISTRO X
       MOV   #$80,DDRD    ;CONFIGURAR EL
      ;PIN 7 DEL PUERTO D COMO SALIDA

INI:
       CLR   PORTD        ;INICIAR PORTD
       BSET  7,PORTD      ;ACTIVO EL BIT
             ;7 DEL PUERTO D. DONDE SE
             ;ENCUENTRA EL LED
       JSR   INI          ;CERRAR EL
             ;CICLO RETORNANDO A INI
        Org  RESET_VEC
        Dw   INICIO       ;AL DARSE
       ;RESET SALTA A INICIO
        END
```

Figura 8.7 *Código del ejercicio 9*

Ejercicio No. 9
Encender y apagar un diodo LED

El objetivo del siguiente ejercicio es continuar con las ideas básicas desarrolladas en el ejercicio anterior. Aprovechando el circuito planteado vamos a hacer titilar un diodo LED a una frecuencia constante. Esto se logra aplicando al diodo un voltaje en forma de onda cuadrada, para lo cual podemos utilizar el temporizador interno e interrupciones o simplemente, como es este caso, por medio de subruitnas de retardo y ciclos de programa.

El circuito de este ejercicio lo podemos montar en una tarjeta de pruebas (*protoboard*), o simplemente lo podemos ejecutar sobre el sistema de desarrollo K-218, el cual, como pudimos apreciar en el proyecto 8, posee un LED de prueba conectado al pin 7 del puerto D.

Lo que se pretende con este ejercicio es generar una señal de nivel alto en el pin 7 del puerto D (PTD7) durante cierto intervalo de tiempo y luego una señal de nivel bajo durante el mismo periodo de tiempo.

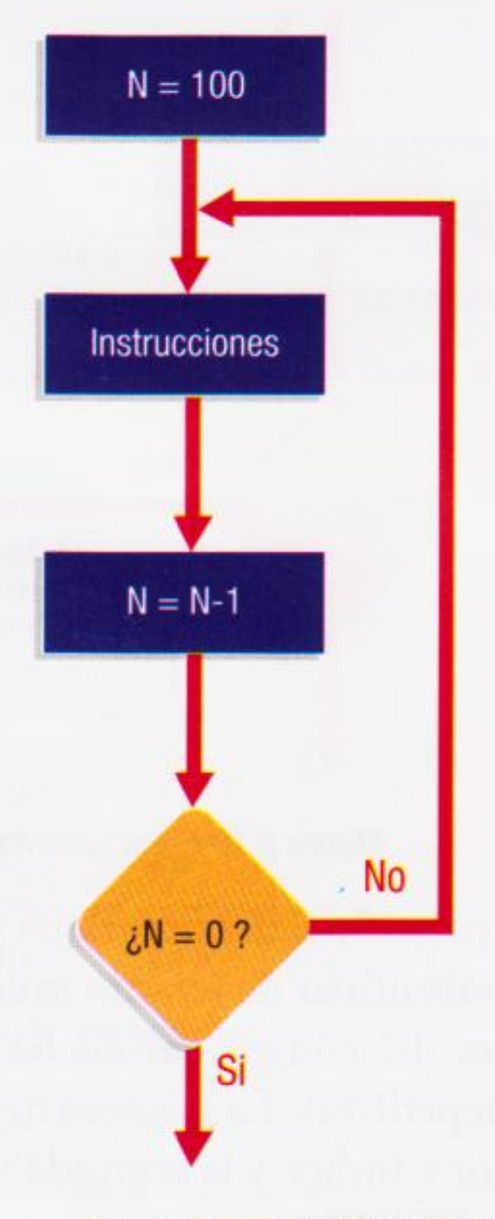


Figura 8.8 *Diagrama de flujo del ejercicio 9*

Para que el ojo humano pueda visualizar el cambio de estado en el LED, debemos hacer que cada nivel de salida permanezca durante un intervalo de tiempo; si no lo hacemos, el microcontrolador ejecutará tan rápido el proceso que no veremos ningún cambio.

Este tiempo de espera es el que hemos denominado **retardo**. Pasado este tiempo la señal aplicada debe pasar a cero. En este punto, si queremos que el proceso se desarrolle de manera repetitiva, basta con ejecutar nuevamente el proceso que se describió en el párrafo anterior. La **figura 8.8** muestra el diagrama de flujo que se acomoda al proceso descrito.

La construcción en lenguaje ensamblador del retardo utilizado se basa en el concepto de estructura repetitiva.

Estructura repetitiva

Este aspecto se refiere a las estructuras cíclicas controladas mediante el uso de una condición lógica. En la **figura 8.8** observamos una secuencia de instrucciones que se ejecutan 100 veces y en donde la condición que determina la termi-

Figura 8.9 *Diagrama de flujo de una estructura repetitiva*

nación del ciclo es el momento en que una variable denominada N llega a cero, después de ser decrementada con cada ciclo ejecutado.

La precaución más importante que se debe tener en cuenta, a la hora de programar estructuras repetitivas, es que se debe crear un mecanismo que permita cambiar la condición inicial que determina el número de veces que se repite el lazo.

En caso de no hacerlo, el programa podría encerrarse en un ciclo infinito, impidiendo que se ejecuten las acciones determinadas por el código escrito a continuación.

Control de estructuras repetitivas

Como hemos visto en varias lecciones anteriores, las estructuras repetitivas por sí mismas en muchas ocasiones no son suficientes para lograr los tiempos que requiere el programa. Para ello recurrimos a anidar ciclos, es decir, incorporar ciclos dentro de ciclos, lo cual contribuye a aumentar el tiempo de retardo generado. En este ejercicio hemos incorporado esta técnica de programación para generar los tiempos de encendido y apagado del LED, el diagrama de flujo genérico para una estructura de este tipo lo podemos visualizar en la **figura 8.9**.

En el código fuente de este ejercicio, **figura 8.10**, podemos distinguir el cuerpo principal del programa, ubicado dentro de la etiqueta INI. Dentro de este bloque se activa el pin 7 del puerto D, se invoca a una subrutina de retardo por medio de la instrucción **JSR RET10S**, donde **RET10S** es la etiqueta que indica el inicio de dicha subrutina.

Una vez terminada la ejecución de la subrutina de retardo, se apaga el LED por medio de la instrucción **BCLR 7, PORTD**, y de nuevo se invoca la subrutina de retardo para que los tiempos de encendido y apagado sean iguales. Finalmente se retorna a INI para que se repita nuevamente el proceso.

Figura 8.10 *Estructuras repetitivas anidadas*

En la parte final del programa podemos observar las subrutinas de retardo utilizadas por las otras partes del código, las cuales emplean estructuras repetitivas. La primera usa los registros acumulador e índice y la segunda utiliza el acumulador y REGS1.

```
*****************************************************************
* Software en assembler para el HC08 JK3. EJERCICIO DE ACTIVACIÓN Y DESACTIVACIÓN
*DE UN LED
*****************************************************************
$include 'jl3regsg.inc'          ; INCLUSIÓN DEL ARCHVIO .INC, DONDE SE ENCUENTRAN
                                 ; LAS DEFINICIONES DE LOS REGISTROS Y BITS DEL
                                 ; MAPA DE MEMORIA
RAMStart     EQU  $0080          ; DIRECCIÓN DE INICIO DE LA MEMORIA RAM
RomStart     EQU  $EC00          ; DIRECCIÓN DE INICIO DE LA MEMORIA ROM
Vectores     EQU  $FFDE
RESET_VEC    EQU  $FFFE          ; UBICACIÓN DEL VECTOR DE RESET

     org RAMStart                ; CON LA DIRETIVA ORG ESTABLECEMOS DENTRO
                                 ; DEL PROGRAMA EL ESPACIO DONDE SE DEFINEN LAS
                                 ; VARIABLES.
CONTADOR        RMB     1        ; A ESTAS TRES VARIABLES SE LE HA RESERVADO UN
REGIS1          RMB     1        ; BYTE DE LA MEMORIA. ELLAS SON UTILIZADAS
REGIS2          RMB     1        ; DENTRO DE LAS SUBRUTINAS DE RETARDO.
     org RomStart                ; AHORA ORG NOS PERMITE INDICAR EL ORIGEN DE LA
                                 ; MEMORIA DE PROGRAMA.
*****************************************************************
*  - DESDE ESTE PUNTO SE INICIA LA EJECUCION DEL CÓDIGO
*             DESPUES DE UN RESET.
*****************************************************************
INICIO:
      RSP
      Bset     COPD,CONFIG1      ; DESHABILITA EL COP (watchdog)
      CLRA                       ; INICIALIZAMOS EL ACUMULADOR EN CERO
      CLRX                       ; INICIALIZAMOS EL REGISTRO X EN CERO (ÍNDICE)
      MOV      #$80,DDRD         ; CONFIGURAMOS EL BIT 7 DEL PUERTO D COMO SALIDA
INI:
      CLR      PORTD             ; INICIALIZAMOS EL PUERTO D EN CERO
      BSET     7,PORTD           ; PONEMOS EN ALTO LA SALIDA DEL BIT 7 DEL PUERTO D
      JSR      RET10S            ; SALTAMOS A LA SUBRUTINA DE RETARDO
      BCLR     7,PORTD           ; UNA VEZ TERMINADO EL RETARDO, APAGAMOS EL LED
      JSR      RET10S            ; Y DE NUEVO INVOCAMOS LA SUBRUTINA DE RETARDO
      JSR      INI               ; EN ESTE PUNTO SE RETORNA A INI PARA EJECUTAR ESTE
                                 ; CICLO DE MANERA INDEFINIDA.
RET10S:                          ; SUBRUTINA DE RETARDO.
      MOV      #$10,CONTADOR     ; INICIALMENTE CARGAMOS EL REGISTRO CONTADOR CON UN
LAZO10 JSR     RETA01            ; DATO, EL CUAL SE DECREMENTA PAULATINAMENTE CADA
      DBNZ     CONTADOR,LAZO10   ; VEZ QUE SE EJECUTA ESTE CICLO DONDE SE INVOCA
      RTS                        ; LA SUBRUTINA RETA01.
*****************************************************************
* RETARDO- SUBRUTINA DE RETARDO POR SOFTWARE
*
*****************************************************************
RETA01                           ; ÉSTA ES LA VERDADERA SUBRUTINA DE RETARDO
        LDA      #128T           ; APROXIMADAMENTE DE 0,1s
ET2     CLRX                     ; AQUI SE CARGA EL ACUMULADOR CON EL NÚMERO
ET3     DECX                     ;  128 Y SE ENTRA A UNA ESTRUCTURA REPETITIVA
        BNE      ET3             ; DONDE DICHO REGISTRO SE VA DECREMENTANDO HASTA
        DECA                     ; LLEGAR A CERO.
        BNE      ET2             ; FINALMENTE SE RETORNA AL PUNTO DESDE DONDE
        RTS                      ; FUE INVOCADA.
RET20S
        LDA     #20T             ; ÉSTA ES OTRA SUBRUTINA DE RETARDO QUE PUEDE
        STA     REGIS1           ; INSERTAR EN EL EJERCICIO. ENSAYE REEMPLAZANDO
ET1     JSR     RETA01           ; LA ANTERIOR SUBRUTINA POR ÉSTA Y OBTENGA UN
        DEC     REGIS1           ; TIEMPO DE RETARDO MAYOR TRABAJANDO CON REGISTROS
        BNE     ET1              ; NORMALES.
        RTS
        Org RESET_VEC            ; EN ESTE CASO LA DIRECTIVA ORG ESTABLECE LO QUE
                                 ; SE DEBE  HACER EN CASO DE PRESENTARSE UN RESET
        Dw INICIO                ; SI ESTE OCURRE, EL PROGRAMA SAlTA INICIO
        END
```

Figura 8.11 *Código fuente del ejercicio*

Simulación dentro del circuito

Ahora vamos a analizar la ventaja que representa simular un programa dentro del circuito, aprovechando las posibilidades que nos brinda el sistema de desarrollo K-218.

Una vez escrito el programa, depurado y ensamblado dentro del entorno WinIDE, ejecutemos el simulador «in circuit» por medio del segundo icono de izquierda a derecha, **figura 8.11**.

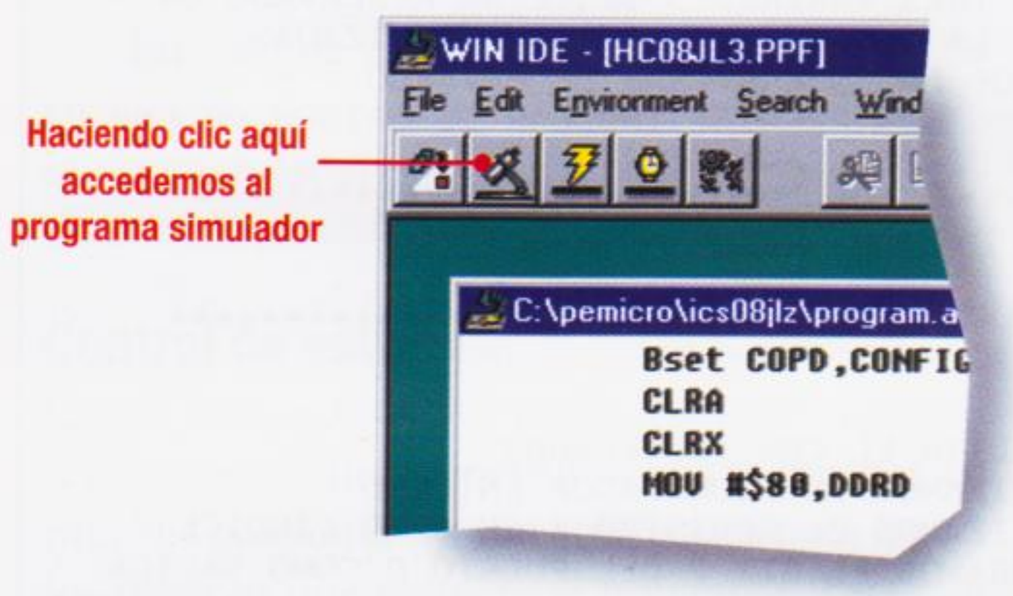


Figura 8.11 *Icono del programa para simular dentro del circuito*

Cuando estamos dentro del programa simulador, **figura 8.12**, tenemos a nuestra disposición una serie de herramientas que nos permiten, por ejemplo, visualizar el valor que toman las variables que intervienen en el programa. Dentro del programa simulador podemos apreciar entre otras, una ventana donde se muestra un mapa de la memoria del microcontrolador y una ventana donde se encuentra el código fuente del programa que se va a someter al proceso de simulación.

Una de las herramientas más importantes con que cuentan los programadores en lenguaje ensamblador es ejecutar los programas paso a paso, esto es, ejecutar una instrucción a la vez. Esto da la posibilidad de observar el comportamiento de todos los registros, variables y estados de la CPU del microcontrolador cada que se ejecuta una línea de código.

Gracias a esta posibilidad se facilita la búsqueda y depuración de errores, sobre todo si, como en este caso, el usuario puede leer señales presentes en los pines del microcontrolador y originar niveles de salida. Todo esto de acuerdo a la configuración que se haga sobre los puertos.

En este programa de simulación podemos ordenar la ejecución de una línea del código (un paso) mediante el quinto icono de izquierda a derecha, o pulsando la tecla F5.

Cuando ejecutamos un programa paso a paso, en la ventana del código de programa se resalta la línea que nos encontramos ejecutando y una flecha se ubica en el lado izquierdo de dicha línea.

Como ejercicio, recomendamos que ejecute paso a paso el programa de este ejercicio y visualice tanto el estado de LED conectado al bit 7 del puerto D, como las variaciones que van tomando las distintas posiciones de memoria y el estado de la CPU.

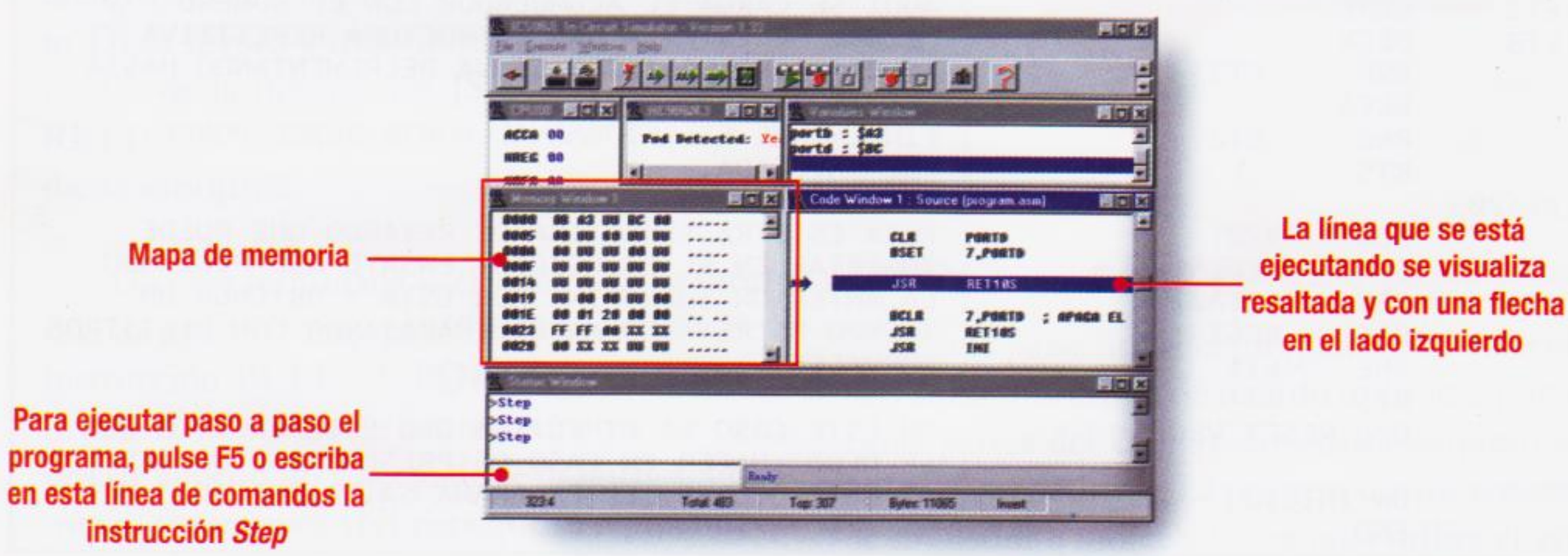


Figura 8.12 *Ejecución paso a paso del programa*

Ejercicio No. 11
Multiplexaje de indicadores (*displays*) de siete segmentos

Para llevar a cabo con éxito los ejercicios propuestos en esta sección y lograr una buena comprensión del manejo del sistema de desarrollo K-218, es recomendable ejecutar los siguientes pasos:

1. Montar el circuito en una tarjeta de pruebas o *protoboard*. Allí se tendrá la posibilidad de evaluar el *hardware*, es decir, hacer una revisión de los circuitos montados para comprobar su buen funcionamiento y así tener la seguridad de que, si existe algún problema, éste debe buscarse en el programa.
2. Escribir el programa en el editor disponible, dentro del entorno WinIDE.
3. Ensamblar el programa corrigiendo errores sintácticos.
4. Simular el programa utilizando el programa simulador del entorno WinIDE (haciendo clic en el icono del microscopio) fuera de línea, es decir, sin la tarjeta conectada. Esto permite verificar que el flujo del programa ha seguido el curso deseado y que los registros se comportan de acuerdo a lo planeado por el programador.
5. Cuando la simulación fuera de línea ha arrojado resultados satisfactorios, es tiempo de ejecutar la simulación dentro del circuito (*in circuit*). Para ello se deben conectar los pines del microcontrolador de la tarjeta K-218 (utilizando los conectores en línea ubicados a lado y lado del microcontrolador JK3) hasta las posiciones que correspondan dentro del *protoboard*, **figura 8.13**. Una vez que el programa de la computadora se ha comunicado correctamente con la tarjeta de acuerdo al análisis desarrollado en el proyecto 8 de este curso, se puede ejecutar nuestro programa en modo paso a paso, o en modo de múltiples pasos (ver comandos STEP, STEPOVER, GO, etc.). El resultado final es que podemos observar el funcionamiento de la aplicación pero a una baja velocidad.

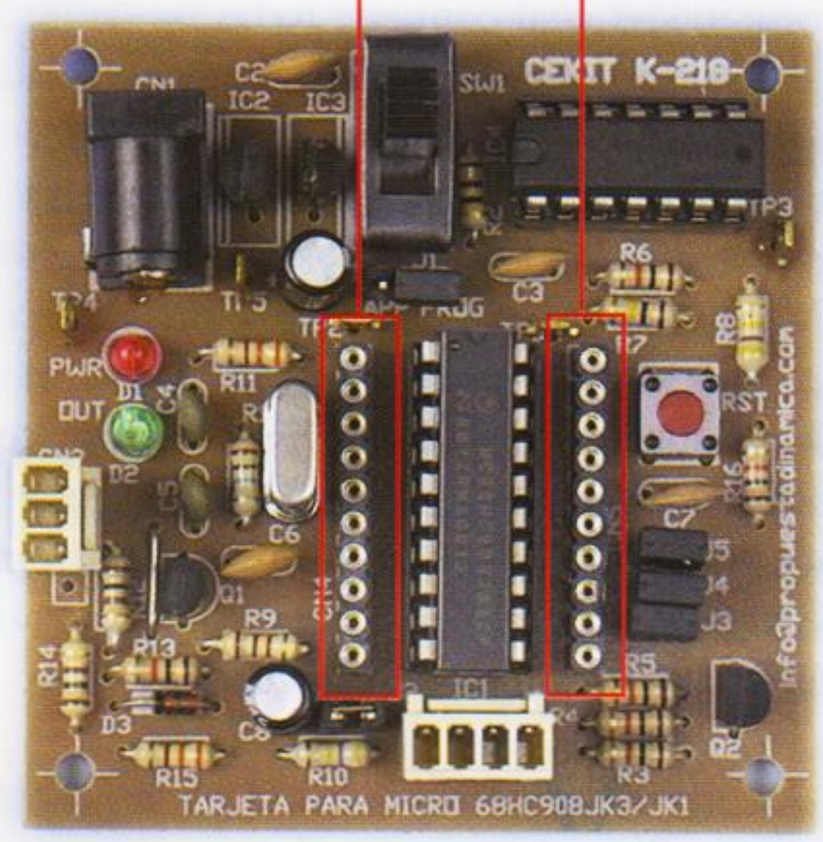


Figura 8.13 *Conectores en línea para cablear hasta el prototipo*

6. Ahora se debe programar el microcontrolador para poder insertarlo de manera definitiva dentro del prototipo y poder visualizar el resultado final en tiempo real.

Ya analizados estos pasos podemos entrar en materia. El manejo de *displays* de siete segmentos constituye una de las técnicas más utilizadas dentro del campo de la programación de microcontroladores, para el desarrollo de equipos electrónicos.

Con estos dispositivos podemos lograr sistemas que permitan al usuario una buena visualización de las distintas variables que se estén manipulando, con la ventaja de ser mucho más económicos que otros medios de visualización, como los *displays* de cristal líquido (LCD). Aunque el principio de funcionamiento con que se lleva a cabo el multiplexaje de *displays* ya lo hemos estudiado en lecciones anteriores, vamos a recordarlo brevemente y a analizar como hacerlo con un microcontrolador Motorola y dos *displays* de cátodo común.

Hemos analizado la estructura básica de un programa en ensamblador para microcontroladores Motorola, **figura 8.14**; podemos notar que siempre empezamos insertando el archivo jl3regsg.inc, el cual contiene todas las definiciones del mapa de memoria del microcontrolador, lo que nos ahorra el tener que escribir las mismas definiciones en cada programa que elaboremos. Luego se definen las direcciones de la memoria RAM y la memoria de programa. Con la directiva org, establecemos primero el inicio de la memoria RAM, lo cual nos permite definir los registros que vamos a utilizar dentro de nuestro código, y segundo el inicio de la memoria de programa; a partir de este punto escribimos el cuerpo del programa o código fuente propiamente dicho.

Dentro de la memoria RAM podemos definir si se utilizan los macros de nuestro programa. Recordemos que los macros son porciones de código que se utilizan repetidamente dentro de un programa, por lo que se encapsulan dentro de un nombre para ser invocados rápidamente y así darle a los programas mayor versatilidad y facilidad de comprensión. Un macro no equivale a una subrutina, la diferencia radica en que, cuando se invoca una subrutina, el contador de programa cambia de valor para dirigir la CPU a determinada posición de memoria y luego retorna a la posición de memoria inicial. Por el contrario, el macro no modifica el contador de programa, simplemente cada que se invoca un macro, incorporamos las líneas de código encapsuladas dentro de éste, al punto exacto del programa desde donde se ha llamado.

Consideremos la estructura de la **figura 8.15**. Allí tenemos dos *displays* de siete segmentos y un que puerto controla dos transistores NPN, que finalmente alimentan los cátodos de cada uno de los *displays*, en donde el pin PTD4 controla el *display* de menor peso.

Los *displays* se conectan a los pines PTB4, PTB5, PTB6 y PTB7 del microcontrolador por medio de un circuito integrado decodificador

```
$include 'jl3regsg.inc'          ;incluye archivo de
                                 ;definiciones

RAM_START          equ $0080     ;dirección de inicio de la
                                 ;RAM
FLASH_START_JK3    equ $EC00     ;only for JK3/JL3

RESET_VEC          equ $FFFE     ;Reset Vector

        org RAM_START

datoDis1     rmb 1               ;separa un byte
datoDis2     rmb 1               ;separa un byte
dispStatus   rmb 1               ;vble de estado para la
                                 ;tarea de visualización
** Estados de la tarea de Visualización****
UPDATE1      equ 0               ;actualizando el display 1
UPDATE2      equ 1               ;actualizando el display 2

* DEFINICION DE MACROS, SI LOS HAY

      org FLASH_START_JK3        ;inicio de la Flash del
                                 ;microcontrolador
```

Figura 8.14 *Encabezado típico de un programa y descripción en bloques de la estructura en un microcontrolador Motorola*

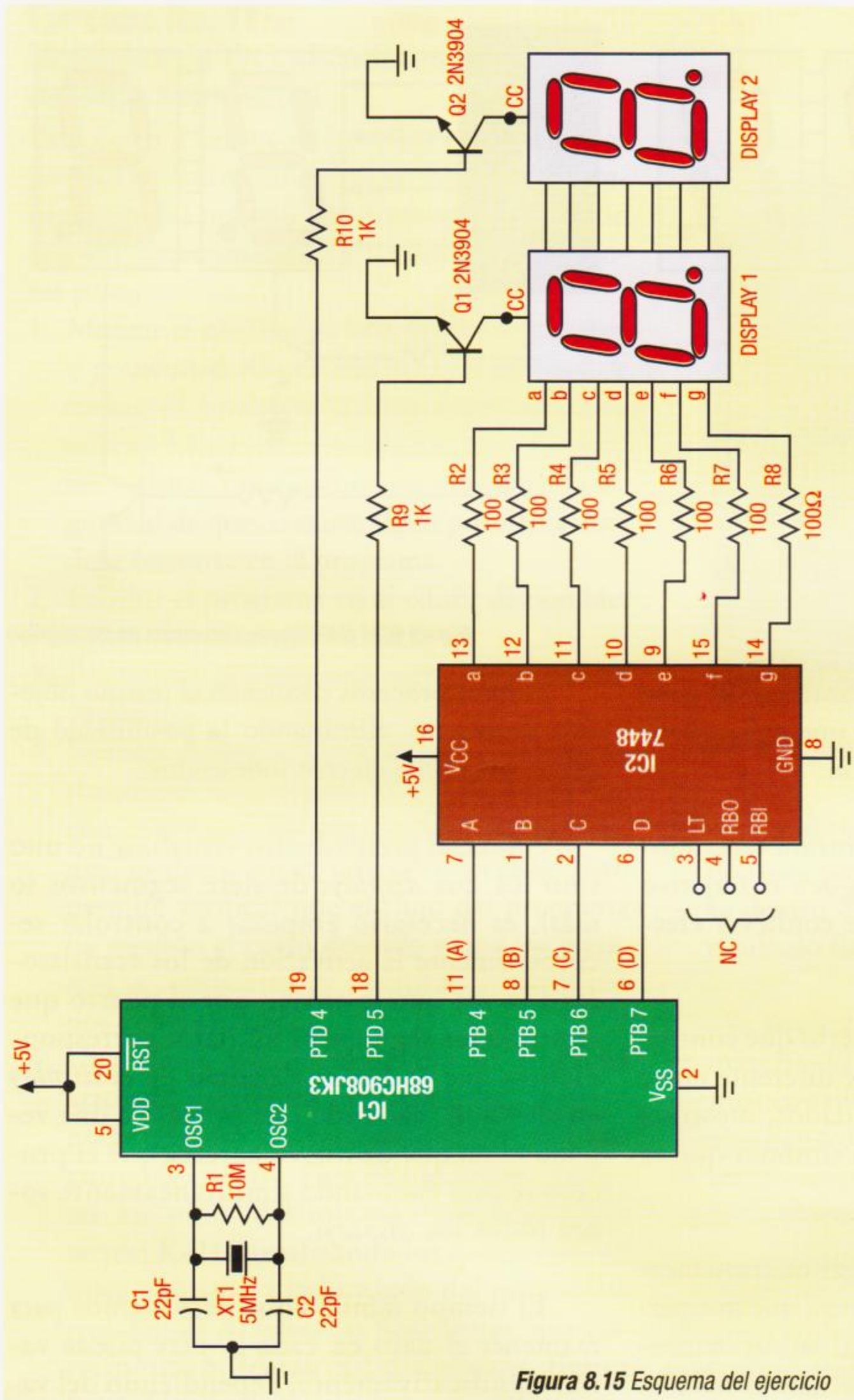


Figura 8.15 *Esquema del ejercicio*

7448; las resistencias R2 a R8 tienen como función limitar la corriente que fluye a través de los segmentos. De esta manera se ahorran pines del microcontrolador que pueden ser útiles para otras funciones dentro del proyecto. Una de las consideraciones más importantes que debemos tener en cuenta dentro del diseño con microcontroladores, es aprovechar al máximo los pines disponibles; para ello debemos hacer uso de algunas técnicas para multiplexar las funciones y los tiempos de uso de cada uno de los pines; sobre este tema hablaremos más adelante.

Observando la configuración del circuito es fácil deducir que, cuando se tiene un nivel lógico bajo en la base del transistor, éste se comporta como un interruptor abierto y no circula corriente entre emisor y colector; se necesita un uno lógico en las bases de éstos para que el transistor se comporte como un interruptor cerrado y se presente dicha corriente.

Si queremos mostrar el número tres en el *display* de la derecha, debemos establecer básicamente los siguientes pasos:

- Activar el transistor conectado al pin PTD4
- Enviar el número 0011 por los pines PTB4, PTB5, PTB6 y PTB7.

El secreto para visualizar adecuadamente un valor numérico en dos o más *displays* de siete segmentos, consiste en habilitar, por medio de un pin del microcontrolador, uno por uno los *displays* de manera independiente y secuencial.

Cada uno de ellos se habilita durante un corto espacio de tiempo, repitiéndose el proceso de manera cíclica, **figura 8.16**.

Los segmentos del *display* deben ir conectados al puerto del microcontrolador directamente o a través de un circuito integrado decodificador.

En el mismo instante en que el primer *display* se en-

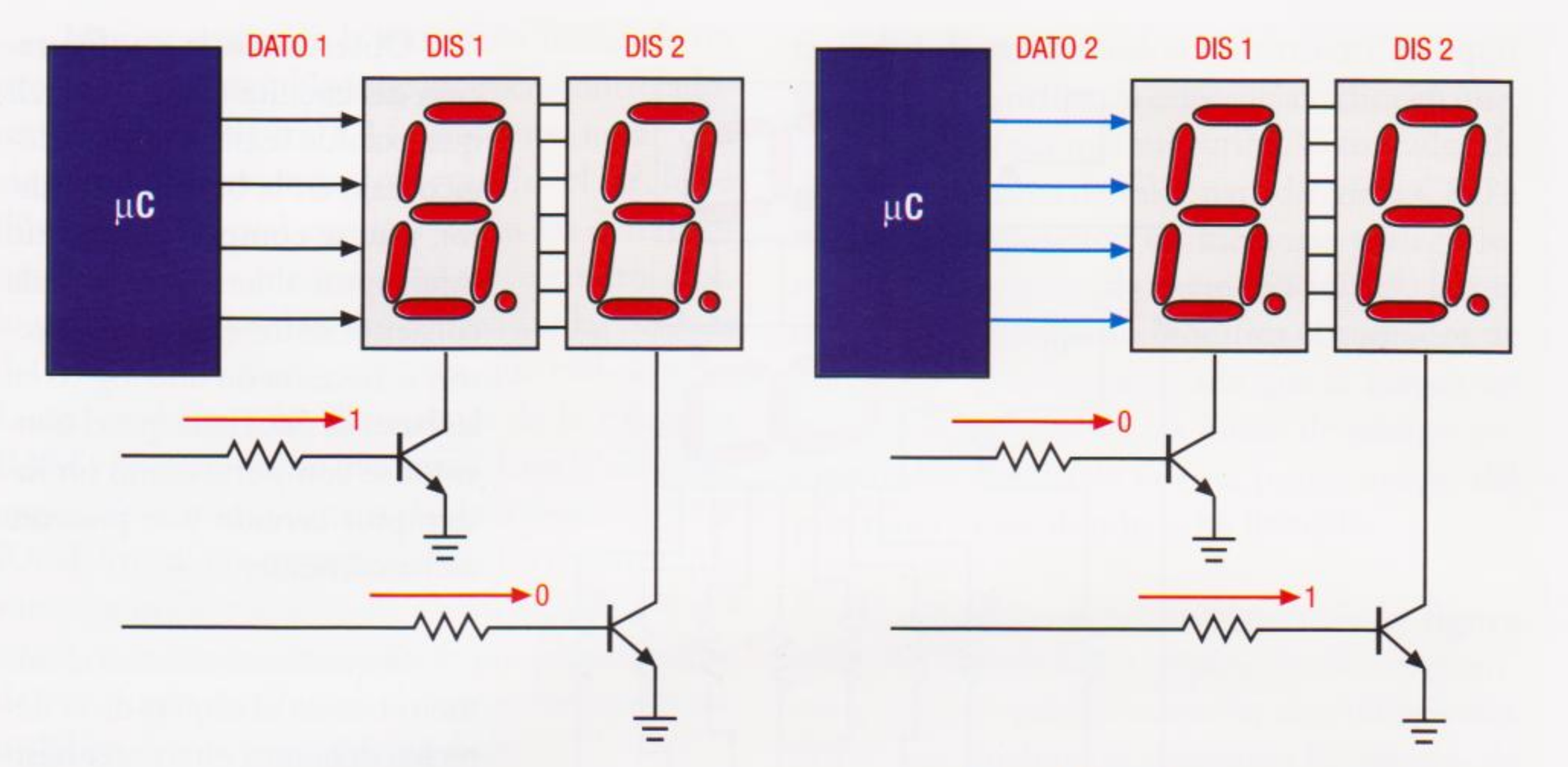


Figura 8.16 *Habilitación secuencial de los displays*

cuentre habilitado, el microcontrolador debe enviar el dato correspondiente que debe ser visualizado en la primera posición.

Un buen observador encontrará que, aunque los anteriores pasos consiguen el objetivo propuesto, este proceso puede conllevar efectos indeseados.

Aquí, por ejemplo, si el puerto que controla los segmentos tenía un valor diferente entre la ejecución de los dos pasos dados, mostrará en el *display* un número o un símbolo que es diferente del valor deseado.

Por ello, es conveniente apagar momentáneamente los segmentos, de tal manera que nos permita seleccionar adecuadamente el *display* en cuestión, y, posterior a esto, enviar el dato correcto a los segmentos.

Otra opción que permite obtener el mismo resultado, con un proceso diferente, sería la siguiente:

- Colocar el número binario 00 en el puerto que controla los transistores
- Enviar el dato al puerto que maneja los segmentos
- Colocar el número binario 01 en el puerto que controla los transistores

Ambos procesos conllevan al mismo objetivo propuesto, eliminando la posibilidad de que se presenten efectos indeseados.

Ahora, si pretendemos visualizar no uno sino los dos *displays* de siete segmentos (o más), es necesario empezar a controlar secuencialmente la activación de los transistores a la vez que se envían por el puerto que controla los segmentos los datos correspondientes que se desean visualizar en el *display* en cuestión, haciendo este proceso a una velocidad tal que de nuevo parezca que el proceso se está ejecutando simultáneamente sobre todos los *displays*.

El tiempo mínimo que necesitamos para mantener el dato en cada *display* puede variar significativamente, dependiendo del valor de las resistencias limitadoras conectadas a cada uno de los segmentos, del número de dígitos que se tengan para mostrar y de la características físicas o eléctricas propias del *display*; experimentalmente se ha encontrado que, mostrar cada dígito durante 3 milisegundos, cuando se tienen resistencias limitadoras de 100 ohmios, proporciona un brillo aceptable y una buena visualización a una distancia prudente.

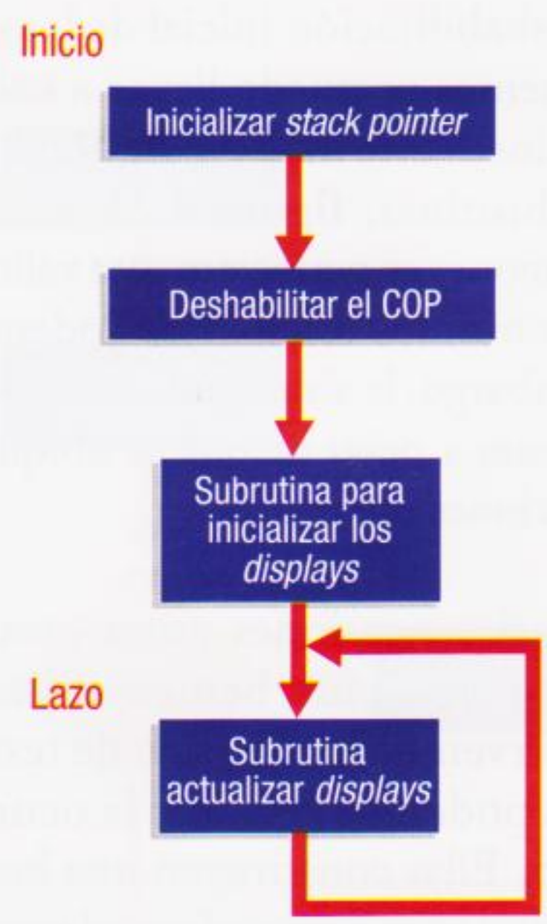


Figura 8.17 *Diagrama de flujo del ejercicio*

```
**** Inicio del programa al dar RESET
*********

Start:
      Rsp                   ;inicializa stack pointer

      Bset   COPD,CONFIG1 ;desabilita el COP
                            ;(watchdog)

      jsr    InitDisplay  ;rutina de inicialización
                            ;de la tarea de display

LAZO                        ;loop principal que
                            ;invoca todas las
                            ;tareas del sistema
      jsr    ActualiceDisplay   ;Tarea de display
      bra    LAZO

******** Inicializa módulo de display
InitDisplay
      clr    datoDis1
      clr    datoDis2
      clr    dispStatus
      bset   UPDATE1,dispStatus
      RTS
```

Figura 8.18 *Código fuente del cuerpo principal del ejercicio*

En la **figura 8.17** podemos apreciar el diagrama de flujo genérico correspondiente al lazo principal que se ejecuta en el presente ejercicio, y en la **figura 8.18** el código fuente en ensamblador de esta porción de programa.

Podemos observar que se trata de una secuencia muy simple; se empieza con dos pasos fundamentales como lo son inicializar el puntero de pila (*stack pointer*) y deshabilitar el perro guardián o COP, estos dos pasos nos permiten iniciar el programa sin correr ningún riesgo, evitando que el COP inicialice constantemente el programa y que el puntero a la pila inicie su recorrido desde cero. En todos los programas de esta sección podremos distinguir la misma estructura.

En primera instancia se ejecuta una subrutina de inicialización de los *displays* don-

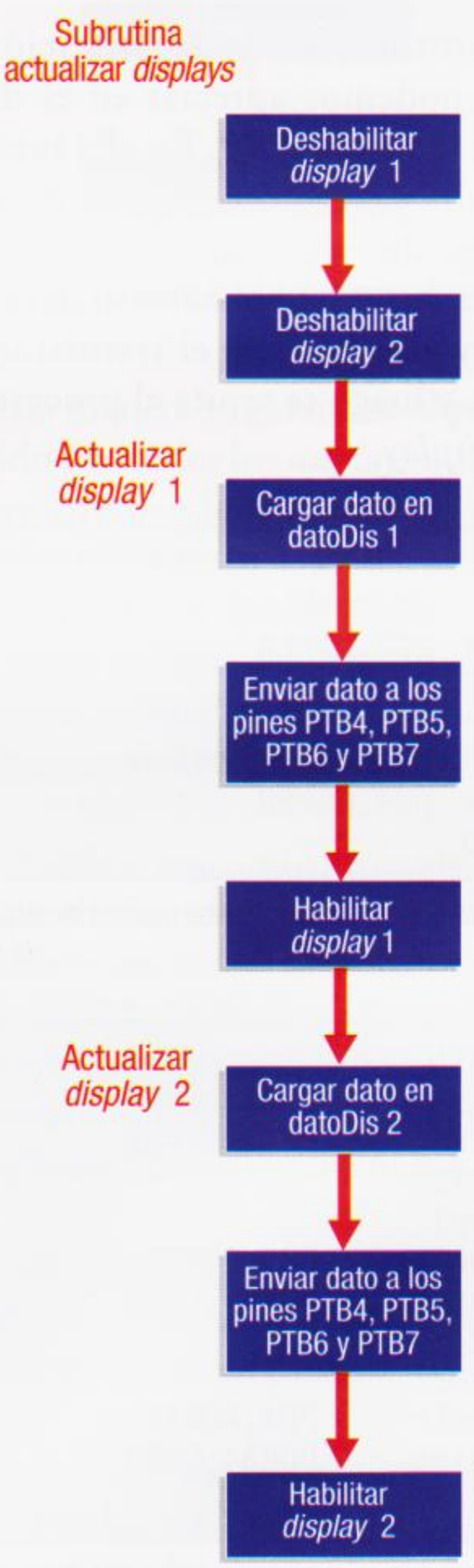


Figura 8.19 *Diagrama de flujo de la subrutina para actualizar los displays*

de fundamentalmente se borran los datos de cada uno de los *displays*; luego, se entra a un ciclo infinito donde se invoca una subrutina de actualización de los *displays*.

En este subrutina se carga el dato que se desea visualizar, ya sea un valor constante como en este caso o un valor que dependa de una entrada externa como un pulsador o un teclado; este caso, al igual que la multiplexación de más *displays,* lo estudiaremos en ejercicios posteriores donde obervaremos como modificar estas subrutinas.

La subrutina para la actualización de los *displays* la podemos apreciar en el diagrama de flujo de la **figura 8.19**. En ella inicialmente se deshabilitan ambos *displays* para no correr el riesgo de visualizar un dato de manera equivocada, luego se actualiza el dato del primer *display* y se habilita el transistor correspondiente, y luego se repite el proceso para el segundo *display*.

```
****DEFINICIÓN DE MACROS*********
$MACRO     deshabDisp1
     bclr  pd5,portd
$MACROEND

$MACRO     deshabDisp2
     bclr  pd4,portd
$MACROEND
```

Figura 8.20 *Definición de los macros de deshabilitación de los displays*

```
DisableAll
        jsr      DisabDisp1
        jsr      DisabDisp2
        RTS
DisabDisp1
        bclr     PD5,PORTD
        bset     DDRD5,DDRD
        RTS
DisabDisp2
        bclr     PD4,PORTB
        bset     DDRB4,DDRB
        RTS
```

Figura 8.21 *Deshabilitación de los displays por medio de subrutinas*

La deshabilitación inicial de los *displays* de siete segmentos se puede llevar a cabo de dos maneras, mediante macros, **figura 8.20**, o mediante subrutinas, **figura 8.21**; cualquiera de las dos opciones es perfectamente válida y deben arrojar los mismos resultados. Podemos observar, sin embargo, la similitud existente entre las dos opciones a pesar de que se ubiquen en distintas porciones del programa.

En varias ocasiones anteriores, en proyectos y en ejercicios, hemos utilizado registros que sirven de banderas o de testigos, con los cuales podemos verificar la ocurrencia de un evento. Ellas constituyen una herramienta fundamental para llevar adecuadamente el flujo del programa, dependiendo de los eventos que se vayan presentando externamente y las condiciones que se originen dentro de la ejecución del programa.

En este caso podemos utilizar un registro bandera llamado **dispStatus** para que nos informe cual *display* se debe actualizar en cada instante del programa; sobre él vamos a definir las siguientes condiciones: cuando el *bit* cero de este registro está en uno, significa que debemos actualizar el *display* uno y, cuando el *bit* uno está en alto, debemos actualizar el *display* dos, **figura 8.22**.

La definición de estos dos *bits* del registro bandera se puede apreciar en el mapa de memoria RAM, donde ellos reciben los nombres de **UPDATE1** y **UPDATE2**. La asignación de etiquetas a números constantes es una manera muy efectiva de poder referirnos fácilmente a determinados *bits* de un registro, en cualquier punto del programa. La facilidad consiste en

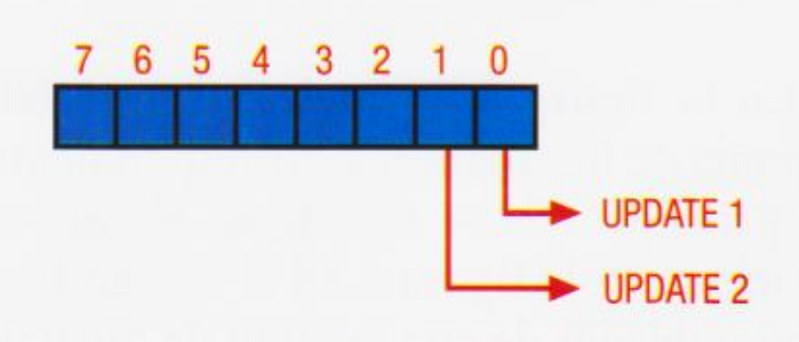


Figura 8.22 *Registro bandera dispStatus*

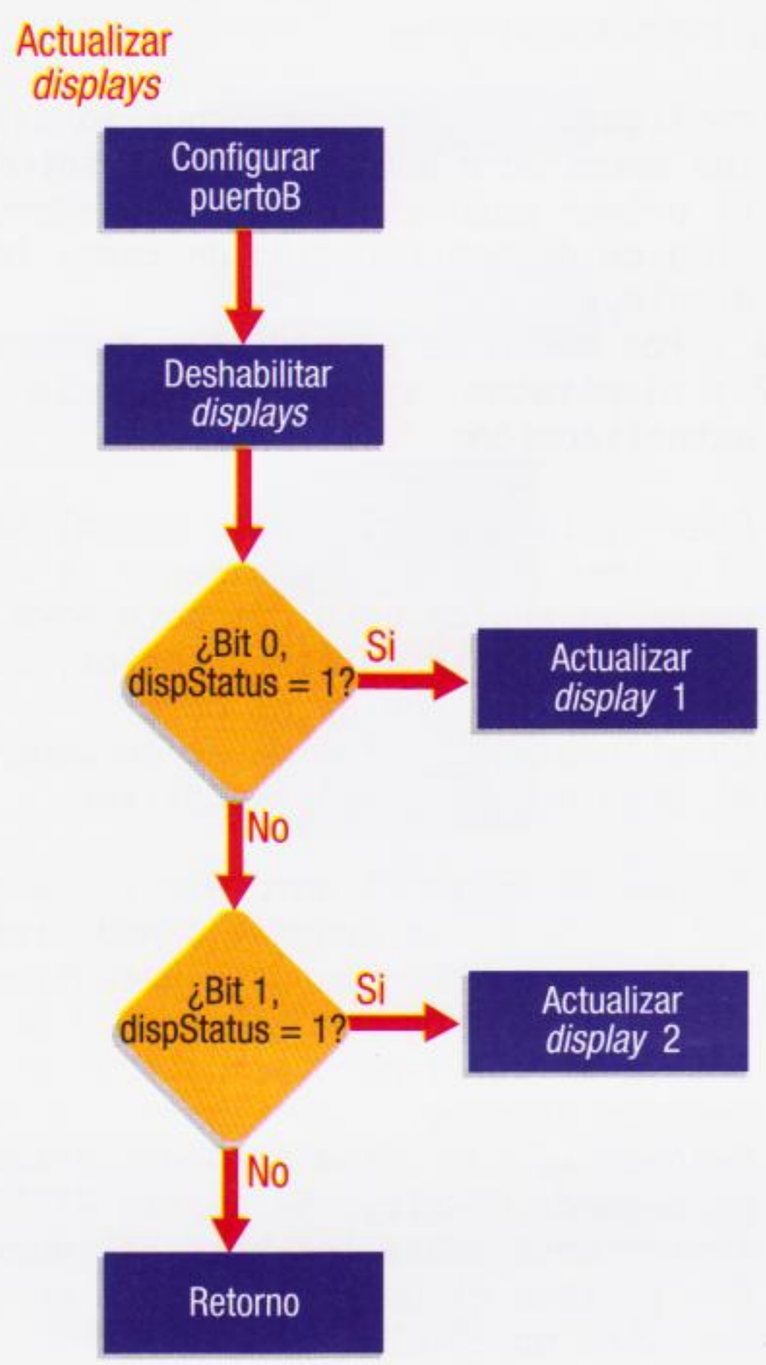


Figura 8.23 *Chequeo de las banderas dentro de la subruitna para actualizar los displays*

que para nosotros es más sencillo recordar un nombre que un número, sobre todo si se tienen varios *bits* que signifiquen algo o varios registros bandera.

Dentro de la subrutina de actualización de los *displays*, el programa pregunta por el estado de dichos *bits* para determinar cuál ya ha sido actualizado y cuál no, **figura 8.23**. Dentro de la subrutina de inicialización de los *displays* se puede observar que se inicializó el *bit* cero en uno para poder empezar a actualizar desde el primer *display*.

Teniendo en cuenta lo anterior, las subrutinas para actualizar cada uno de los *displays* quedan como se muestra en la **figura 8.24**. La diferencia radica en que una vez terminada la actualización del primer *display*, el *bit* cero del registro **dispStatus** es borrado y el *bit* uno es activado.

Figura 8.24 *Actualización del primer display*

De esta manera se garantiza el poder hacer el recorrido por todos los *displays* utilizando los *bits* del registro bandera. El método de actualización de los *displays* utilizado en este caso facilita la incorporación de un mayor número de *displays*, como veremos más adelante. El código fuente de esta subrutina de actualización lo podemos apreciar en la **figura 8.25**.

Un objetivo casi indispensable para toda persona que decida entrar al mundo del desarrollo electrónico con microcontroladores, es estandarizar la mayor cantidad de procedimientos y funciones que hayan sido lo suficientemente probados, dentro de librerías que puedan ser insertadas o incluídas dentro de una nueva aplicación.

Con esto garantizamos que el tiempo de desarrollo de un producto se reducirá significativamente, a la vez que se asegura una buena calidad.

Las subrutinas de manejo multiplexado de *displays* como las que se han analizado en este ejercicio, son un claro ejemplo de una función cuyo código se puede estandarizar.

```
******** Hace la actualización sobre el display correspondiente
ActualiceDisplay
      mov       #$F0,PORTB                    ; configuración del puerto que maneja
      mov       #$F0,DDRB                     ; los datos para que sirvan como salidas
      jsr       DisableAll                    ; El primer paso es saltar a la subru-
                                              ; tina de deshabilitación de todos los
                                              ; displays
      brset     UPDATE1,dispStatus,UpdateDisp1 ; Por medio de el registro bandera
      brset     UPDATE2,dispStatus,UpdateDisp2 ; dispStatus, sigo una secuencia de
      RTS                                     ; actualización.

UpdateDisp1                                   ; Ésta es la subrutina para actualizar
       lda      PORTB                         ; el primer display. Incialmente se debe
       and      #$0F                          ; hacer un tratamiento del dato para
       sta      PORTB                         ; enviarse por los bits correctos, es
       inc      datoDis1                      ; decir los que envían el dato.
       lda      datoDis1                      ; En el registro datoDis1 se encuentra
       nsa                                    ; el dato que se desea visualizar.
       ora      PORTB
       sta      PORTB                         ; Cuando se envía el dato por los pines
       jsr      EnableDisp1                   ; del puerto B, se procede a habilitar
       bclr     UPDATE1,dispStatus            ; el display. Ahora se debe desactivar
       bset     UPDATE2,dispStatus            ; el bit 0 del registro bandera y se
       RTS                                    ; activa el bit 1 para actualizar el
UpdateDisp2                                   ; segundo display.
       lda      PORTB                         ; Ésta es la subrutina para actualizar
       and      #$0F                          ; el segundo display. Se repite el
       sta      PORTB                         ; tratamiento sobre los bits del puerto
                                              ; B y se toma el dato almacenado en el
       lda      datoDis2                      ; registro datoDis2.
       nsa
       ora      PORTB
       sta      PORTB
       jsr      EnableDisp2                   ; Ahora se salta a la subrutina de
       bclr     UPDATE2,dispStatus            ; habilitación del segundo display y
       bset     UPDATE1,dispStatus            ; se hacen los respectivos cambios
       RTS                                    ; sobre el registro bandera.
DisableAll
       jsr      DisabDisp1                    ; la subrutina para deshabilitar los
       jsr      DisabDisp2                    ; displays invoca dos nuevas subrutinas
       RTS                                    ; una para cada uno de ellos.
EnableDisp1
       bclr     PD5,PORTD                     ; Para habilitar cada uno de los
       bset     DDRD5,DDRD                    ; displays, basta activar el
       RTS                                    ; transistor conectado al cátodo común
EnableDisp2                                   ; de cada uno.
       bclr     PD4,PORTD
       bset     DDRD5,DDRD
       RTS
DisabDisp1                                    ; De la misma manera, para deshabilitar
       bset     PD5,PORTD                     ; cada uno de los displays, basta
       bset     DDRD5,DDRD                    ; desactivar el transistor asociado
       RTS                                    ; a cada uno de los cátodos.
DisabDisp2
       bset     PD4,PORTD
       bset     DDRD4,DDRD
       RTS
 Org RESET_VEC                                ; fin del programa
 Dw Start   ;al darse reset salta a Start
```

Figura 8.25 *Código fuente de la subrutina para actualizar los displays*

Ejercicio No. 12
Manejo del conjunto de instrucciones

En la sección de teoría hemos analizado diversos aspectos relacionados con la arquitectura, el modelo de programación y los modos de direccionamiento en los microcontroladores Motorola. Por medio de tablas mostramos todo el conjunto de instrucciones al igual que su funcionamiento básico y cantidad de ciclos de máquina empleados por cada una. Siempre que vayamos a cambiar de tecnología o de marca de microcontroladores, tal vez el paso más crítico es entender y manejar los modos de direccionamiento para poder llevar a cabo cualquier operación de programación.

En este ejercicio estudiaremos algunos ejemplos específicos de manejo de las instrucciones, con los cuales nos podemos familiarizar rápidamente con la programación de este tipo de microcontroladores. Empezaremos con el movimiento de información entre posiciones de memoria, es decir, modos de direccionamiento, instrucciones de salto y acceso a tablas, entre otros.

Enfatizaremos en el modo de ejecutar cada uno de los ejemplos en la familia HC05 y en la familia HC08, para que podamos visualizar las diferencias existentes y las ventajas que nos ofrece esta última.

Modos de direccionamiento

Para hacer un seguimiento de cada uno de los ejemplos, debemos remitirnos al código fuente de la **figura 8.26**, en el cual se resaltan cada uno de los numerales que se explican a continuación.

```
*****************************************************************
*                                                               *
*      ESTE ARCHIVO MUESTRA EJEMPLOS PARA LA INSTRUCCIÓN MOV    *
*       SE COMPARAN ALGUNOS MODOS DE DIRECCIONAMIENTO ENTRE     *
*      MICROCONTROLADORES MOTOROLA 05 Y 08.                     *
*                                                               *
*          CURSO PRÁCTICO SOBRE MICROCONTROLADORES              *
*               CEKIT - PEREIRA - COLOMBIA                      *
*****************************************************************


        ORG     $200

*****   A. MOVER UN VALOR A UNA POSICIÓN DE MEMORIA

          *       CÓDIGO PARA HC05                    *
INICIO    LDA        #$30            ;2,2 ACCA <- $30 : EL ACUMUL, SE CARGA
                                     ;                  CON EL NÙMERO $30
          STA        $80             ;4,2 ($80) <- (ACCA), PASA EL CONTENIDO
A                                    ; A LA POSICIÓN DE MEMORIA 80
          *       CODIGO PARA HC08                    *
          MOV        #$30,$80        ;4,3 ($80) <- $30

*----------------       CICLOS EN UNA CPU05, BYTES      = 6,4  --------------
*----------------       CICLOS EN UNA CPU08, BYTES      = 4,3  --------------

*****   B. MOVER UN VALOR DE UNA POSICIÓN DE MEMORIA A OTRA

B         *       CÓDIGO PARA HC05                    *
          LDA        $80             ;3,2 ACCA <- ($80) ;MOVER EL CONTENIDO DE
                                     ;LA POSICIÓN 80 AL ACUMULADOR
```

```
            STA       $90              ;4,2 ($90) <- (ACCA); MOVER EL ACC. A LA
                                       ;POSICIÓN 90.

B           *      CODIGO PARA  HC08                    *
            MOV       $80,$90          ;5,3 ($90) <- ($80), MOVER LOS
                                       ; CONTENIDOS DE LA POSICIÓN DE MEMORIA
                                       ; 80 A LA POSICIÓN DE MEMORIA 90.

*----------------      CICLOS EN UNA CPU05, BYTES      = 7,4 ---------------
*----------------      CICLOS EN UNA CPU08, BYTES      = 5,3 ---------------
```

```
*****   C. MOVER LOS CONTENIDOS DE UNA POSICIÓN DE MEMORIA
*****INDEXADA A UNA POSICIÓN DIRECTA. LUEGO INCREMENTAR REGISTRO X

            *  CÓDIGO PARA HC05                        *
            LDX       #$80             ;    X <- $80
            LDA       ,X               ;3,1 ACCA <- (X)
            STA       $90              ;4,2 ($90) <- (ACCA)
C           INCX                       ;3,1 X <- X + 1
                                       ; EN ESTE PROCESO SE UTILIZAN 10 CICLOS

            *  CÓDIGO PARA HC08                        *
            LDX       #$80             ;    X <- $80
            MOV       X+,$90           ;4,2 ($90) <- (X)
                                       ;    X <- X + 1
                                       ; EN ESTE PROCESO SE UTILIZAN 4 CICLOS
* ---------------      CICLOS EN UNA CPU05, BYTES      = 10,4 ---------------
* ---------------      CICLOS EN UNA CPU08, BYTES      =  4,2 ---------------
```

```
*****   D. MUEVE LOS CONTENIDOS DE UNA POSICIÓN DIRECTA DE MEMORIA A UNA
*****       POSICIÓN INDEXADA , LUEGO INCREMENTA X

            *      CÓDIGO PARA  HC05                    *
            LDX       #$90             ;    X <- $90
            LDA       $80              ;3,2 ACCA <- ($80)
            STA       ,X               ;4,1 (X) <- (ACCA)
D           INCX                       ;3,1 X <- X + 1

            *      CÓDIGO PARA  HC08                    *
            LDX       #$90             ;    X <- $90
            MOV       $80,X+           ;4,2 (X) <- ($80)
                                       ;    X <- X + 1

* ----------------     CICLOS EN UNA CPU05, BYTES      = 10,4 --------------
* ----------------      CICLOS EN UNA CPU08, BYTES     = 4,2 ---------------
```

```
*****   INICIALIZAR VECTOR DE RESET
            ORG       $FFFE
            DW         INICIO
```

Figura 8.26 *Código de distintos modos de direccionamiento*

A. Movimiento de un valor numérico hacia una posición de memoria

Para llevar un valor fijo a una posición de memoria, en un microcontrolador HC05, se debe hacer la operación mediante el acumulador siguiendo dos pasos: primero se carga el dato en el acumulador con la instrucción LDA y luego, mediante la instrucción STA, se lleva el dato a la posición de memoria final. En este proceso se emplean seis ciclos de máquina.

En un microcontrolador HC08 este proceso se desarrolla rápidamente debido a la capacidad que tienen estos últimos de direccionamiento de una posición inmediata a una posición directa. Simplemente escribiendo MOV #$30,$80 llevamos el número 30 a la posición de memoria 80, empleando sólo 4 ciclos de máquina, **figura 8.27**. Este tipo de direccionamiento es muy utilizado para asignar valores a registros de configuración. Por ejemplo, para establecer los valores del registro CONFIG, podemos utilizar este tipo de direccionamiento escribiendo MOV #$01,CONFIG1.

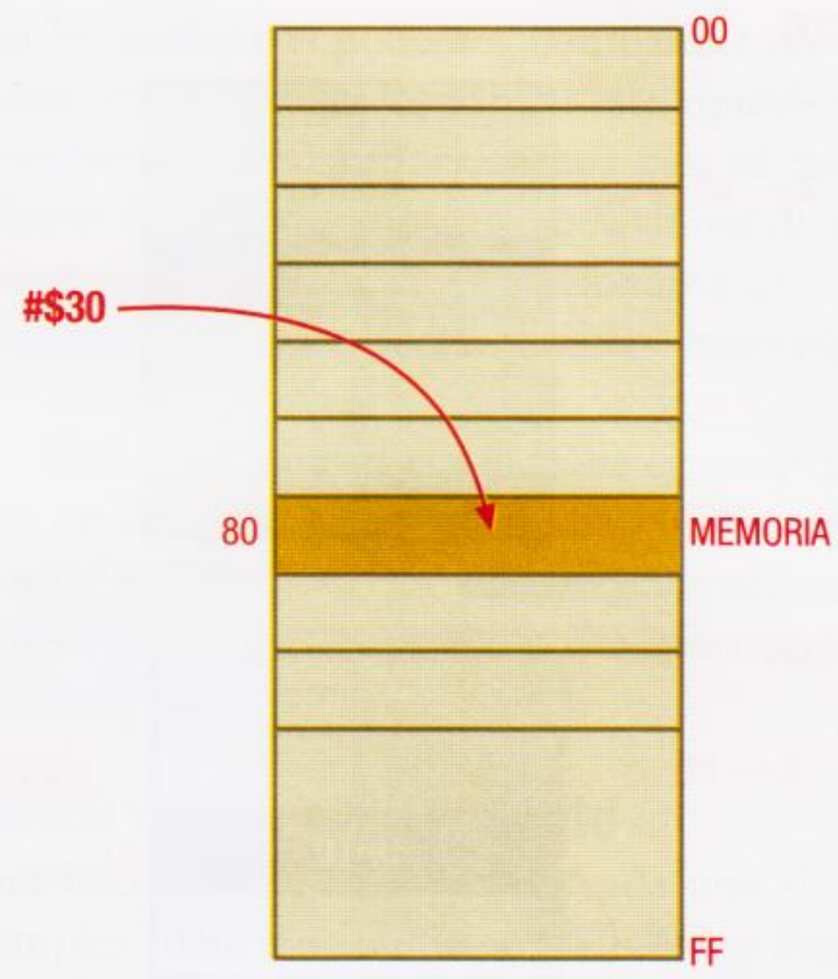


Figura 8.27 *Representación del movimiento de un valor a una posición de memoria*

B. Movimiento de un valor desde una posición de memoria a otra

En el caso de los microcontroladores HC05 el procedimiento es similar al anterior pues se manipula el acumulador para ejecutar la tarea, pero en este caso se emplean siete ciclos de máquina. Un microcontrolador Motorola permite hacer este movimiento en un solo paso gracias al direccionamiento de una posición directa a otra posición directa. Ésta es una de las características más llamativas que presentan estos microcontroladores y que los hacen tan poderosos.

C. Mueve los contenidos de una posición de memoria indexada a una posición directa. Luego incrementa registro X.

Recordemos que las instrucciones que utilizan el modo de direccionamiento indexado, se caracterizan por acceder a datos localizados en posiciones de memoria variables. El registro X contiene el *byte* bajo de la dirección y el registro H el *byte* alto (esto para los microcontroladores HC08 que poseen el registro H:X). Gracias a esta posibilidad, los microcontroladores HC08 no se limitan a los primeros 256 *bytes* de memoria como ocurre con los HC05.

Este tipo de direccionamiento es muy práctico para acceder a tablas localizadas en memoria, ya que el registro X puede incrementarse permitiendo desarrollar un barrido desde una dirección determinada. En la familia HC05 se deben ejecutar cuatro pasos como vemos en este ejemplo:

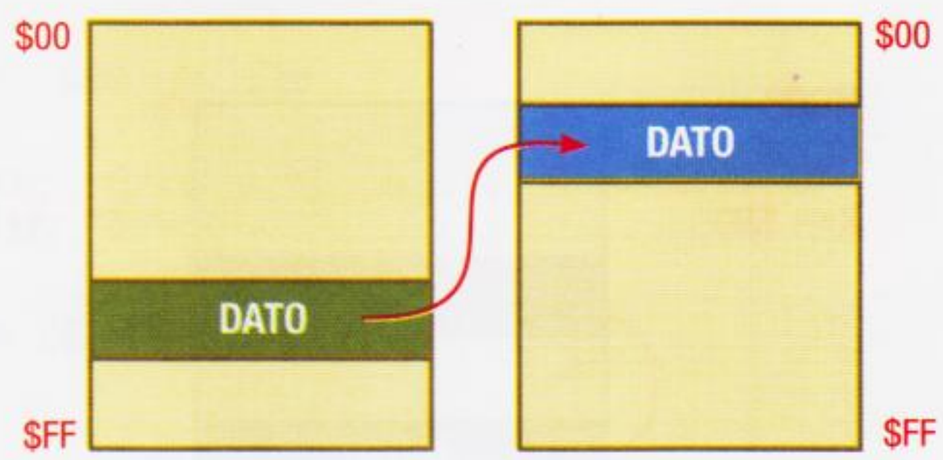


Figura 8.28 *Movimiento de datos entre dos posiciones de memoria*

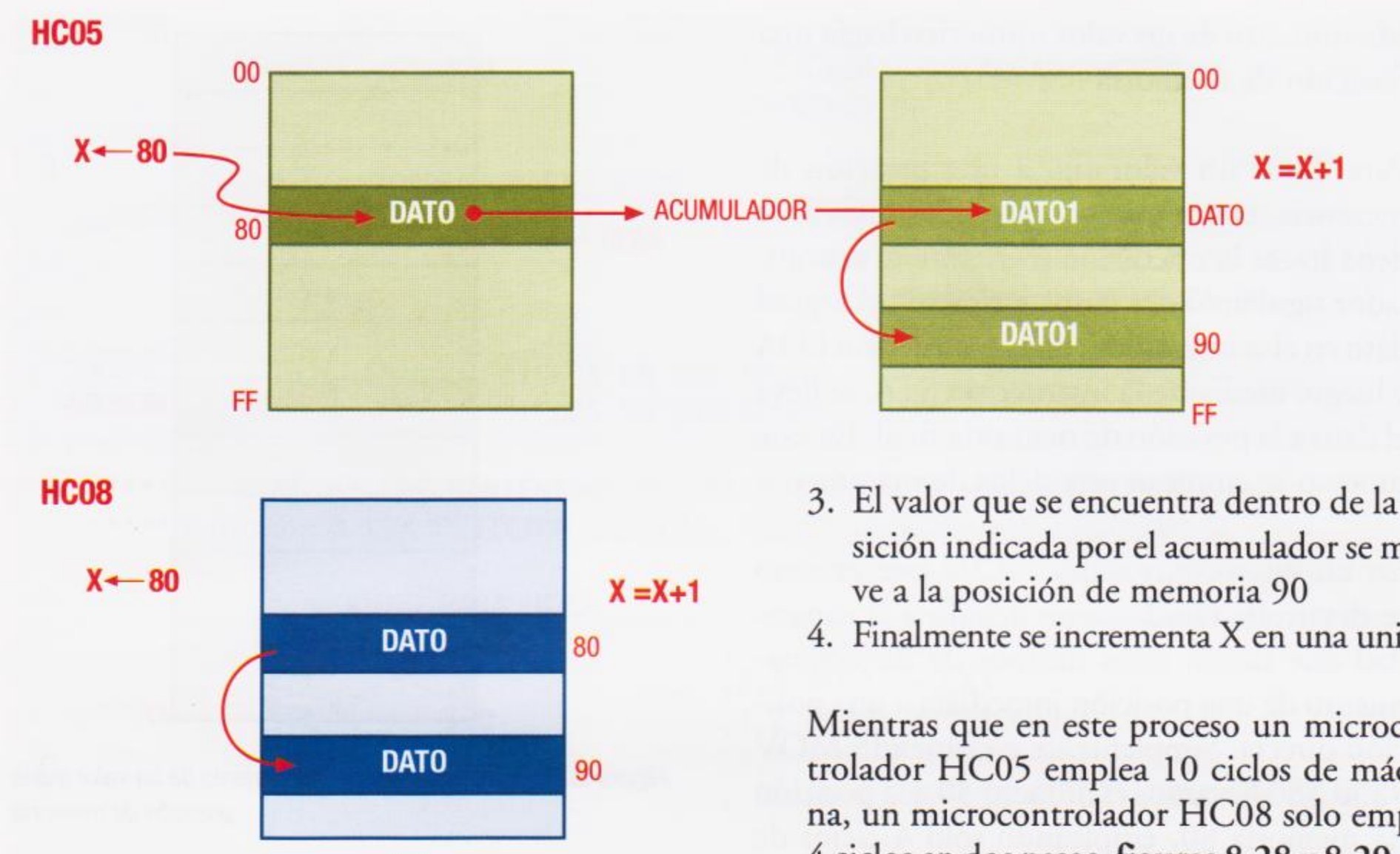


Figura 8.29 *Comparación del movimiento de datos entre posiciones de memoria, entre los microcontroladores HC05 y HC08*

1. Se carga el registro índice X con el número 80
2. El valor que se encuentra en la posición de memoria indicada por X se pasa al acumulador por medio de la instrucción LDA
3. El valor que se encuentra dentro de la posición indicada por el acumulador se mueve a la posición de memoria 90
4. Finalmente se incrementa X en una unidad

Mientras que en este proceso un microcontrolador HC05 emplea 10 ciclos de máquina, un microcontrolador HC08 solo emplea 4 ciclos en dos pasos, **figuras 8.28** y **8.29**, así:

1. Se carga el registro índice X con el número 80
2. Haciendo uso del direccionamiento que permite el movimiento de los contenidos de una posición de memoria indexada a una posición directa con incremento del registro X, se puede escribir MOV X+,$90 y de esta manera el valor contenido en la posición de memoria indicada por el registro X se mueve a la posición indicada dentro de la dirección 90 e inmediatamente se incrementa el registro X.

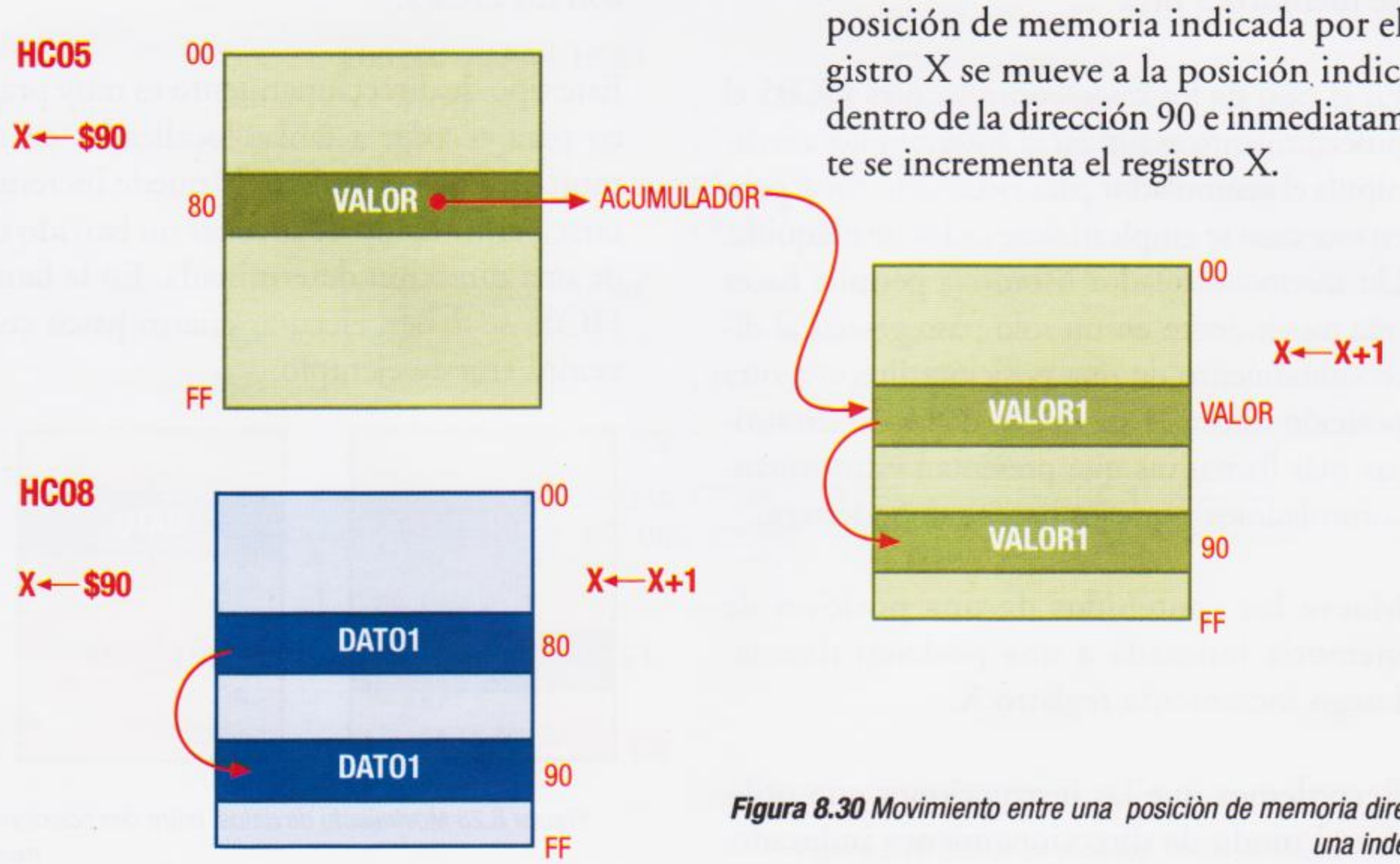


Figura 8.30 *Movimiento entre una posiciòn de memoria directa a una indexada*

D. Movimiento de los contenidos de una posición directa de memoria a una posición indexada, luego incrementar X.

Este modo de direccionamiento es recíproco al anterior. En este caso el contenido de una posición de memoria se mueve a la posición indicada por el registro X. En los microcontroladores HC05 se deben ejecutar cinco pasos que emplean diez ciclos de máquina, **figura 8.30**, así:

1. Se carga el registro X con el número 90
2. El valor almacenado en la pósición 80 se carga en el acumulador mediante la instrucción LDA $80
3. La posición indicada por el valor contenido dentro del acumulador se lleva a la posición indicada por el registro X
4. Se incrementa X en una unidad

En la familia HC08, gracias a sus poderosos métodos de direccionamiento, el proceso se reduce a dos pasos que consumen sólo cuatro ciclos de máquina, así:

1. Se carga el registro índice X con el número 90
2. Aprovechando la capacidad de direccionamiento de los HC08, se ejecuta la instrucción MOV $80,X+, la cual mueve el contenido de la posición 80 a la posición indicada por X

Instrucciones de salto

Ahora vamos a analizar algunas nuevas instrucciones de salto de los microcontroladores HC08 que no están presentes en el HC05. Por medio de los ejemplos del código de la **figura 8.31** ilustraremos las ventajas de estas nuevas instrucciones; aprenderemos su sintaxis y la manera de incorporarlas en nuestros algoritmos.

a. CBEQ – Compara los valores contenidos en dos posiciones de memoria y salta si ellos son iguales.

Esta instrucción se cataloga dentro del grupo de direccionamiento directo, el cual se caracteriza por acceder directamente a posiciones de memoria.

En este primer caso el acumulador se carga con $40 y la posición de memoria $80 contiene $40. En el microcontrolador HC05 se deben ejecutar tres pasos que emplean seis ciclos de máquina. Como podemos apreciar en el código fuente, por medio de la instrucción **CMP $80** se hace la resta del contenido del acumulador, del valor contenido en la posición 80. En un microcontrolador de la familia HC08 basta utilizar la instrucción CBEQ $80,LPB, la cual resta el contenido de la posición de memoria señalada por el acumulador y el contenido de la posición de memoria señalada dentro la posición de memoria $80. Si el resultado de la resta es cero, salta a la etiqueta LPB.

b. CBEQA - Comparar y saltar si es igual.

Esta instrucción se cataloga como de direccionamiento inmediato, el cual se caracteriza por llevar a cabo la acción con el dato que tiene a continuación. Si comparamos el código del HC05 de este caso con el anterior, observamos que la instrucción CMP #$50 ejecuta una resta entre el contenido del acumulador y el valor numérico $50; allí es donde se encuentra la naturaleza del direccionamiento inmediato.

Para llevar a cabo esta tarea, un microcontrolador HC05 emplea tres líneas de código y cinco ciclos de máquina. Si el contenido del acumulador es $50 entonces se salta a la etiqueta LP2, y si no lo es, entonces retorna a LPB; estas dos condiciones se establecen por medio de las instrucciones BEQ y BRA.

En el caso del microcontrolador HC08, el código lo podemos apreciar a partir de la etiqueta LP2. Simplemente se utiliza la instrucción CBEQA la cual hace directamente la comparación entre el valor numérico $50 y el valor indicado por el acumulador, y salta a la etiqueta LPC en caso de confirmar la igualdad entre dichos valores. En este caso, para desarrollar la

```
*****************************************************************************
*                                                                           *
*       En este programa se muestran algunas instrucciones de               *
*       salto de la CPU08 como CBEQ, CBEQA, CBEQX                           *
*                              DBNZ, DBNZA, DBNZX                           *
*                                                                           *
*                CURSO PRÁCTICO SOBRE MICROCONTROLADORES - CEKIT            *
*****************************************************************************
        ORG     $200

*****   CBEQ - compara y salta si es igual (direccionamiento directo) ******************
*       el acumulador se pre-define como $40                                *
*       La posición de memoria $80 contiene $40                             *
*****************************************************************************
    *       HC05                    *
LPA         CMP         $80                     ;3,2 (ACCA) - ($80)
            BEQ         LP1                     ;3,2 si (ACCA) = ($80) entonces
                                                ;salta a LP1
            BRA         LPA                     ;va a LPA, intente de nuevo!

    *       HC08                    *
LP1         CBEQ        $80,LPB                 ;5,3 si (ACCA)-($80)=0,
                                                ;entonces salta a LPB
            BRA         LP1                     ;va a LP1

*       Total de ciclos de la CPU05, bytes      = 6,4
*       Total de ciclos de la CPU08, bytes      = 5,3

*****   CBEQA - compara y salta si es igual (direccionamiento immediato)******************
*       el acumulador se predefine en $50                                   *
*****************************************************************************
*       HC05                    *
LPB         CMP         #$50                    ;2,2 (ACCA) - $50
            BEQ         LP2                     ;3,2 si (ACCA) = $50, entonces LP2
            BRA         LPB                     ;ir a LPB

*       HC08                    *
LP2         CBEQA       #$50,LPC                ;4,3 si #$50 = (ACCA), entonces LPC
            BRA         LP2                     ;ir a LP2

*       Total de ciclos de la CPU05, bytes      = 5,4
*       Total de ciclos de la CPU08, bytes      = 4,3

*****   CBEQX - compara y salta si es igual (direccionamiento indexado)*******************
*       El registro índice X se predefine en $60                            *
*****************************************************************************
*       HC05                    *
LPC         CPX         #$60                    ;2,2 X - $60
            BEQ         LP3                     ;3,2 si X = $60, entonces LP3
            BRA         LPC                     ;ir a LPC

*       HC08                    *
LP3         CBEQX       #$60,LPD                ;4,3 si X = $60, entonces LPD
            BRA         LP3                     ;ir a LP3

*       Total de ciclos de la CPU05, bytes      = 5,4
*       Total de ciclos de la CPU08, bytes      = 4,3

*****   DBNZ - decremente y salte si no es cero ******************************************
*****************************************************************************
*       HC05                    *
*       La posición de memoria $A0 se predefine en $08
LPD         NOP                                 ;usado aquí para representar cualquier
                                                número de instrucciones
            DEC         $A0                     ;5,2 decrementa ($A0)
            BNE         LPD                     ;3,2 si ($A0) no es cero, entonces LPD

*       HC08                    *
*       La posición de memoria $A0 se predefine en $08
LP4         NOP                                 ;usado aquí para representar cualquier
                                                ;número de instrucciones
```

```
            DBNZ        $A0,LP4                 ;5,3 ($A0) = ($A0) - 1
                                                ;si ($A0) no es cero, entonces LP4

*       Total de ciclos de la CPU05, bytes        = 8,4
*       Total de ciclos de la CPU08, bytes        = 5,3

*****   DBNZA - decremente el acumulador y salte si no es cero*****************************
*******************************************************************************************
*       HC05                    *
*       El acumulador se predefine en $06
LPE         NOP                                 ;usado aquí para representar cualquier
                                                ;número de instrucciones
            DECA                                ;3,1 (ACCA) = (ACCA) - 1
            BNE         LPE                     ;3,2 si (ACCA) no es cero, entonces LPE

*       HC08                    *
*       El acumulador se predefine en $06
LP5         NOP                                 ;usado aquí para representar cualquier
                                                ;número de instrucciones
            DBNZA       LP5                     ;3,2 (ACCA) = (ACCA) - 1
                                                ;si (ACCA) no es cero, entonces LP5

*       Total de ciclos de la CPU05, bytes        = 8,4
*       Total de ciclos de la CPU08, bytes        = 3,2

*****   DBNZX - decremente x y salte si no es cero*****************************************
*******************************************************************************************
*       HC05                    *
*       el registro índice X se predefine en $04
LPF         NOP                                 ;usado aquí para representar cualquier
                                                ;número de instrucciones
            DECX                                ;3,1 (X) = (X) - 1
            BNE         LPF                     ;3,2 si (X) no es cero, entonces LPF

*       HC08                    *
*       el registro índice X se predefine en $04
LP6         NOP                                 ;usado aquí para representar cualquier
                                                ;número de instrucciones
            DBNZX       LP6                     ;3,2 (X) = (X) - 1
                                                ;si (X) no es cero, entonces LP6

*       Total de ciclos de la CPU05, bytes        = 6,3
*       Total de ciclos de la CPU08, bytes        = 3,2

DONE        NOP
            BRA     DONE

*****   Inicializar el vector de reset
            ORG         $FFFE
            DW          LPA
```

Figura 8.31 *Código fuente*

tarea solo se utilizaron dos líneas de código y cuatro ciclos de máquina.

c. CBEQX – Comparar y saltar si es igual mediante direccionamiento indexado. Recordemos que este tipo de direccionamiento utiliza el registro índice X en el HC05, y el H:X en el HC08, para acceder a posiciones de memoria variables. Para ello se utiliza X como registro índice que apunta a cada posición que se desea acceder. Por medio de esta instrucción se hace una comparación con el registro X para determinar cuando debe terminar el barrido o recorrido de una serie de posiciones de memoria.

Para este ejemplo vamos a definir previamente el registro X en $60. En el código del HC05 (etiqueta LPC) se compara X con el número $60 por medio de la ins-

trucción CPX $60; si son iguales, se salta a LP3, de lo contrario va a LPC. En el HC08 el proceso se simplifica con la instrucción CBEQX $60,LPD con la cual se hace directamente la comparación y se salta a LP3 si se detecta la igualdad entre X y el número $60.

d. DBNZ – Decrementar y saltar si no es cero. Con esta instrucción se hace un decremento en una unidad del valor contenido en una posición de memoria. Este proceso es muy útil en una gran variedad de casos de programación, como por ejemplo en la creación de retardos. Para ejecutar satisfactoriamente este ejemplo se debe cargar previamente la posición de memoria $A0 con el valor $08.

En un microcontrolador de la familia HC05 el proceso se hace como se muestra en el código ubicado en la etiqueta LPD. Con la instrucción DEC $A0 se ejecuta el decremento del valor almacenado en la posición A0 empleando cinco ciclos de máquina; para determinar cuando llega a cero ese valor se debe utilizar la instrucción BNE LPD, la cual hace retornar a LPD mientras la cantidad sea distinta de cero.

En un microcontrolador de la familia HC08 el proceso se reduce significativamente mediante la instrucción DBNZ, la cual, además de ejecutar la resta, determina cuando ese resultado es cero y salta dependiendo de esa decisión.

e. DBNZA – Decrementar el acumulador y saltar si no es cero. El registro acumulador es uno de los más importantes en la arquitectura de cualquier microcontrolador, pues es el que más manipulaciones sufre durante la escritura de un programa. Esta instrucción permite decrementar directamente el acumulador e indagar si ha llegado a cero.

De acuerdo a esa determinación, la instrucción permite hacer un salto a otra parte del programa. Para seguir el código fuente vamos a asumir que el acumulador se ha inicializado previamente con el valor $06; este caso es uno de los más notables en cuanto se tiene una reducción significativa de los ciclos de la CPU entre una CPU05 y una CPU08, gracias a la enorme riqueza de esta última en cuanto a modos de direccionamiento.

Si nos remitimos a la etiqueta LPE del programa, observaremos como se lleva acabo esta tarea en un microcontrolador HC05. Se requiere de dos instrucciones: la primera, para decrementar el acumulador (DECA), la cual utiliza tres ciclos de máquina, y, la segunda (BNE), para determinar cuando ha llegado a cero, la cual utiliza otros tres ciclos de máquina.

A partir de la etiqueta LP5 tenemos el código correspondiente al HC08, el cual, como se puede ver, solo consta de una línea DBNZA LP5. Esta instrucción decrementa el acumulador en una unidad, si no ha llegado a cero salta a LP5, todo esto empleando sólo tres ciclos de máquina.

f. DBNZX – Decrementar X y saltar si no es cero. Esta instrucción permite manipular el registro índice X facilitando al usuario el direccionamiento de diversas posiciones de memoria. Para desarrollar este ejercicio se define inicialmente el registro índice con el valor $04. En una arquitectura HC05 se emplea el doble de ciclos de máquina y se utilizan dos instrucciones: la primera, es DEC X con la que se le resta una unidad al registro X, y, la segunda es BNE LPF con la que se determina si X es cero, y en caso negativo salta a LPF.

En una CPU08 solo se requieren tres ciclos de máquina utilizando la instrucción DBNZX LP6. Esta línea se puede traducir como: restar una unidad al registro X y si no ha llegado a cero, saltar a LP6.

Instrucciones varias y manipulación del CCR

Ahora desarrollaremos algunos ejemplos con instrucciones tales como la división, el manejo del BCD y la manipulación del registro CCR. Gracias a que podemos descargar gratuitamente el programa WinIDE de la página www.pemicro.com, podemos simular todos estos ejercicios, a la vez que modificamos desde la ventana de comandos algunos registros, como el acumulador, el CCR, etc. Por ejemplo, para cargar el acumulador con el número 37 simplemente se debe escribir A 37 en la ventana de comandos.

Las instrucciones con las cuales trabajaremos en esta ocasión son: DAA (ajuste decimal del acumulador), NSA (intercambio de *nibbles* en el acumulador), DIV (dividir), TAP (transferir el acumulador al CCR), TPA (transferir el CCR al acumulador). El manejo de todas estas instrucciones las podemos apreciar en el código fuente de la **figura 8.32**.

A. Instrucción DAA

Esta instrucción ajusta el contenido del acumulador al igual que el estado del *bit* de acarreo (*carry*) del registro CCR, después de que se ha ejecutado alguna operación que tenga que ver con números binarios codificados en decimal (BCD). De esta manera se puede sumar correctamente en BCD con una indicación apropiada del *carry*.

En el ejemplo A del código fuente del programa, podemos apreciar que inicialmente se carga el acumulador con el valor en hexadecimal $26, luego se hace una operación de suma con el valor en hexadecimal $37 mediante la instrucción ADD #$37.

El resultado es el número en hexadecimal 5D, el cual se almacena de nuevo en el acumulador. Este resultado es posible trasladarlo a formato BCD por medio de la instrucción DAA. El resultado final que nos proporciona esta instrucción no es el traslado directo de 5D a formato BCD, sino el resultado de sumar los dos números como si estuvieran en formato BCD, lo que finalmente nos da 63. En otras palabras, el valor hexadecimal 5D se ha ajustado al valor en BCD 63.

B. Instrucción NSA

Esta instrucción ejecuta un intercambio de posición de los *nibbles* del registro acumulador. Mientras, para cumplir con esta tarea en un microcontrolador de la familia HC05 se deben desarrollar nueve instrucciones para hacer este intercambio, en un microcontrolador HC08 basta ejecutar la instrucción NSA.

Para ver el desarrollo de este ejemplo debemos cargar con un valor el acumulador para observar como finalmente los *nibbles* de este valor se intercambian. Donde inicia el código del HC05 escribimos A 37 en la ventana de comandos, y ejecutamos paso a paso (pulsando F5) cada una de las instrucciones.

Una vez terminada la ejecución del código del HC05 y antes de iniciar el código para el HC08 carguamos de nuevo el acumulador con $37 (A 37), y, simplemente ejecutando NSA, observaremos el mismo resultado que se obtuvo con todas las instrucciones utilizadas en el HC05.

C. Instrucción DIV

Esta instrucción es capaz de desarrollar una división de un número de 16 *bits* entre un número de 8 *bits*. En el ejemplo que se muestra en el código fuente, se ha conformado un número de 16 *bits* uniendo los registros H y el acumulador, los cuales se dividen entre el número cargado en el registro índice X. El registro H se ha cargado con $02, el registro X se ha cargado con $80 y el acumulador con $00. El número de 16 *bits* que se ha formado H:A es 0200 y el de 8 *bits* X es $80. De esta manera la operación resultante es 0200/80, cuyo resultado es $04.

```
*************************************************************
*                                                           *
*          EJEMPLO DEL MANEJO DE LAS INSTRUCCIONES          *
*          DAA, NSA, DIV, TAP, TPA DE LA CPU08              *
*          SE INCLUYE EL MANEJO BCD, DIVISIÓN,              *
*          Y OPERACIONES CON EL CCR                         *
*          CURSO PRÁCTICO SOBRE MICROCONTROLADORES - CEKIT  *
*          PEREIRA - COLOMBIA - 2001                        *
*************************************************************
; EJECUTE PASO A PASO ESTE PROGRAMA DESDE EL SIMULADOR DEL ENTORNO WINIDE

      ORG    $EC00

***** A.   DAA - AJUSTE DECIMAL DEL ACUMULADOR

START     LDA      #$26          ; ACCA <- $26, UN NÚMERO BCD
          ADD      #$37          ; ACCA <- $37 + (ACCA)
A                                ; ACCA = $5D, UN NÚMERO HEXADECIMAL
          DAA                    ; (ACCA) = 63 = (26 + 37)
                                 ; EL NÚMERO HEXADECIMAL 5D SE HA
                                 ; AJUSTADO AL NÚMERO EN BCD 63

***** B.  NSA - INTERCAMBIO DE NIBBLES EN EL ACUMULADOR
*        EL ACUMULADOR SE DEFINE PREVIAMENTE CON $37
*        AL FINAL EL ACUMULADOR QUEDA CON $73

          ; *     CODIGO PARA EL HC05  *
          TAX                    ;2,1 X <- (ACCA)
          ROLX                   ;3,1 ROTAR X A LA IZQUIERDA
B         ROLA                   ;3,1 ROTAR EL ACC. A LA IZQUIERDA
          ROLX                   ;3,1 ROTAR X A LA IZQUIERDA
          ROLA                   ;3,1 ROTAR EL AC. A LA IZQUIERDA
          ROLX                   ;3,1 ROTAR X A LA IZQUIERDA
          ROLA                   ;3,1 ROTAR EL ACC. A LA IZQUIERDA
          ROLX                   ;3,1 ROTAR X A LA IZQUIERDA
          ROLA                   ;3,1 ROTAR EL ACC. A LA IZQUIERDA
                                 ;CADA INSTRUCCIÓN ROLA O ROLX EMPLEAN
                                 ;3 CICLOS DE MÁQUINA LO QUE DA UN TOTAL
                                 ;DE 26 CICLOS EN LA CPU05
          ; *     CÓDIGO PARA EL HC08  *
          NSA                    ;3,1 INTERCAMBIAR LOS NIBBLES DEL ACC.

; **       TOTAL DE CICLOS EN LA CPU05, bytes       = 26,9
; **      TOTAL DE CICLOS EN LA CPU08, bytes       = 3,1

***** C.  DIV - DIVIDIR 16 BITS POR 8 BITS
*        EL MODO DE DIRECCIONAMIENTO INMEDIATO SE USA PARA CARGAR LOS REGISTROS
*        PARA ILUSTRAR LOS COMPONENTES NECESARIOS PARA EJECUTAR
*         LA INSTRUCCIÓN DIV.

          LDHX     #$0200        ; H <- $02
C         LDX      #$80          ; X <- $80
          LDA      #$00          ; ACCA <- $00
          DIV                    ; H:A / X = A residuo en H
                                 ; LA RESPUESTA ES $04 residuo 0
```

```
***** D.  TAP - TRANSFERIR EL ACUMULADOR AL CCR
*        EL ACUMUL. SE PREDEFINE CON $E2
*        CCR = %0110,0000

D        TAP                          ; CCR <- (ACCA)
                                      ; CCR = %1110,0010

***** E.  TPA - TRANSFERIR EL CCR AL ACUMULADOR
*        ACCA SE PREDEFINE CON $00
*        CCR = %1110,0010

E        TPA                          ; ACCA <- (CCR)
                                      ; ACCA = $E2

DONE     NOP
         BRA       DONE

*****    INICIALIZAR EL VECTOR DE RESET
         ORG          $FFFE
         DW      START
```

Figura 8.32 *Código fuente de los ejercicios de manejo de instrucciones*

D. Instrucción TAP

La función de esta instrucción es transferir el acumulador al registro CCR. Para hacer un buen seguimiento a este ejemplo veamos los siguientes pasos:

1. Ejecutando paso a paso todo el programa, ubiquemos el cursor en la línea de la instrucción TAP, **figura 8.33**.
2. Carguemos el acumulador con el valor $E2, desde la ventana de comandos.

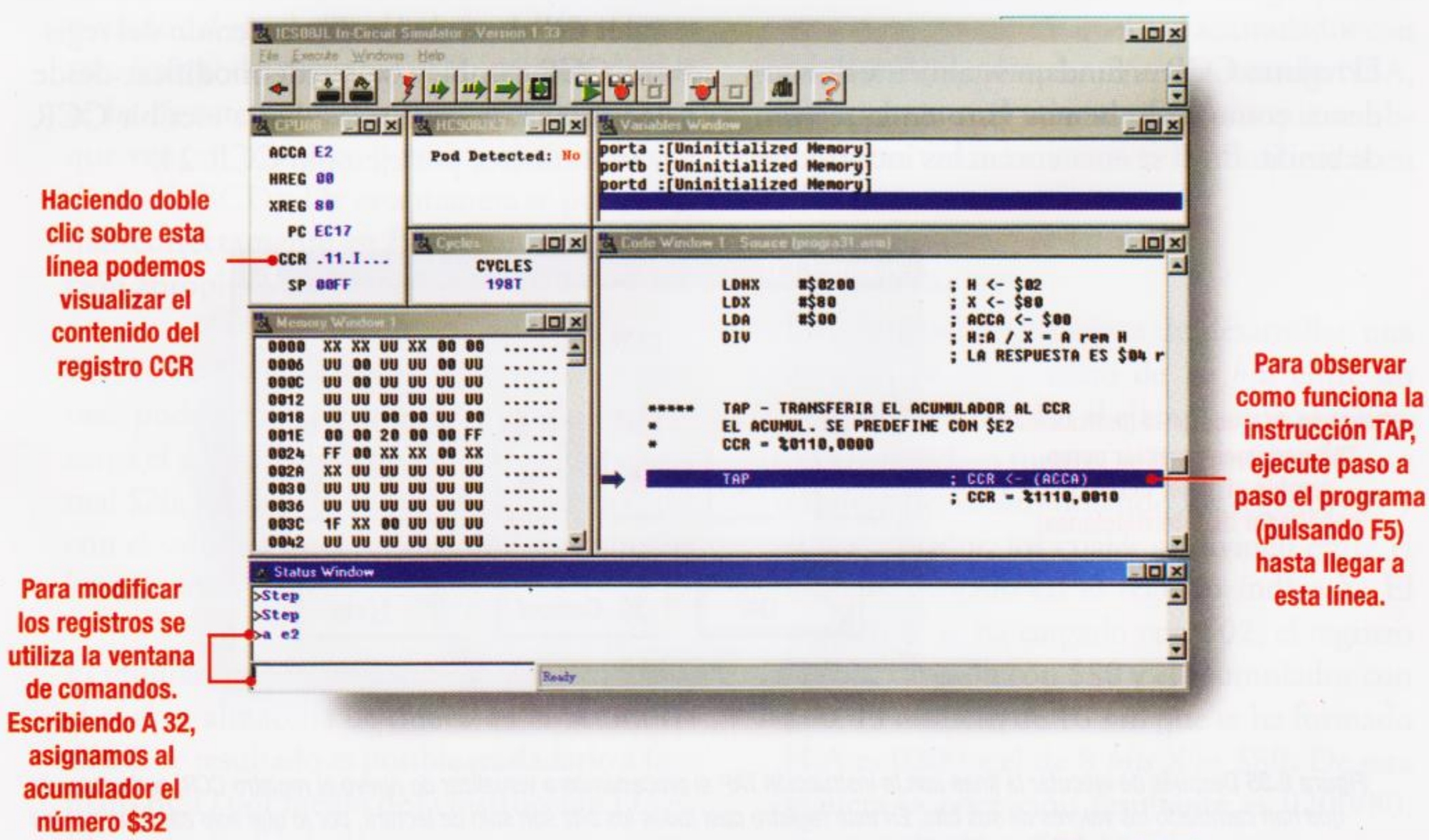


Figura 8.33 *Programa de simulación*

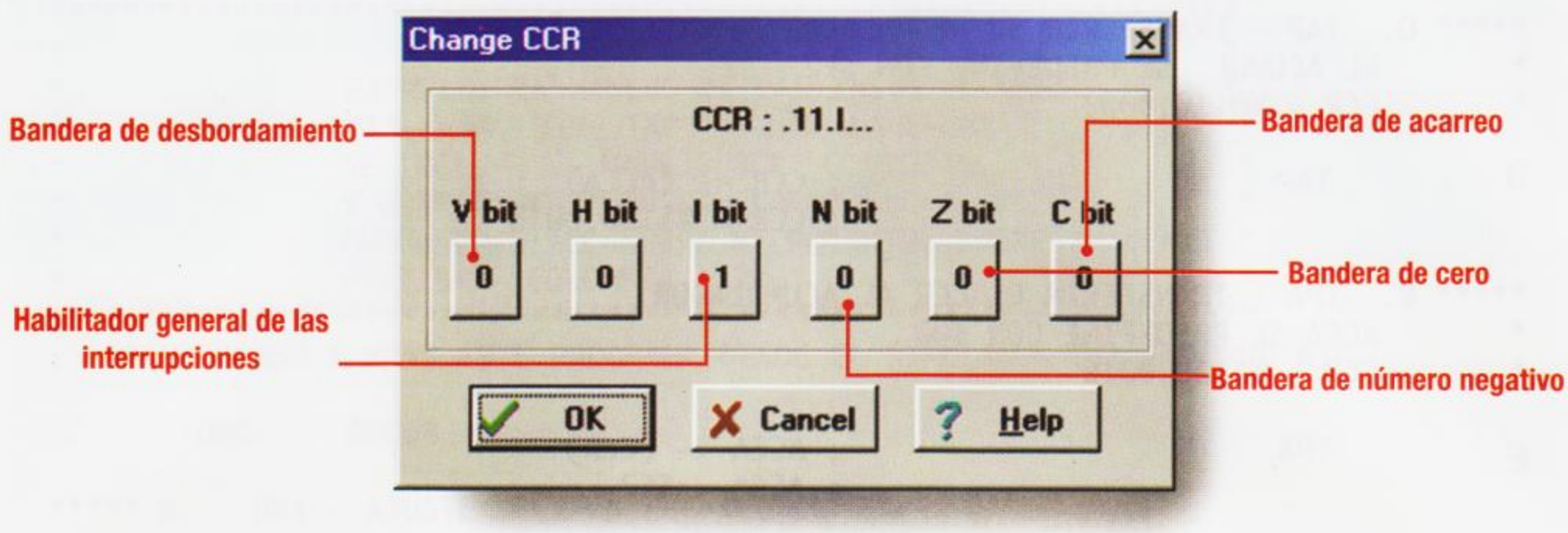


Figura 8.34 *Esta es la visualización a la que tenemos acceso desde el programa simulador. En esta nueva ventana podemos apreciar cada uno de los bits que constituyen el registro CCR, con su respectivo valor.*

3. Haciendo doble clic sobre el nombre del registro CCR, ubicado en la ventana de registros de la CPU08, se visualiza el con tenido actual de éste, **figura 8.34**. Luego se cierra esta ventana.
4. Ahora, pulsemos F5 para ejecutar la instrucción TAP, la cual modifica los contenidos de CCR. Ahora abramos de nuevo la ventana del CCR y observemos los resultados, **figura 8.35**.

El registro CCR es fundamentalmente de banderas, como ya lo hemos visto en la sección de teoría. En él se encuentran los indicadores de cero, *carry*, etc. Sin embargo existe el *bit* I el cual actúa como habilitador global de interrupciones.

E. Instrucción TPA

Esta instrucción es recíproca a la anterior. En este caso se transfiere el contenido del registro CCR al acumulador. Para visualizar el efecto de esta instrucción, pulsemos F5 para ejecutarla y observemos como ha cambiado el valor del acumulador. El contenido del registro CCR también se puede modificar desde la ventana de comandos, basta escribir CCR y un número, por ejemplo CCR 24.

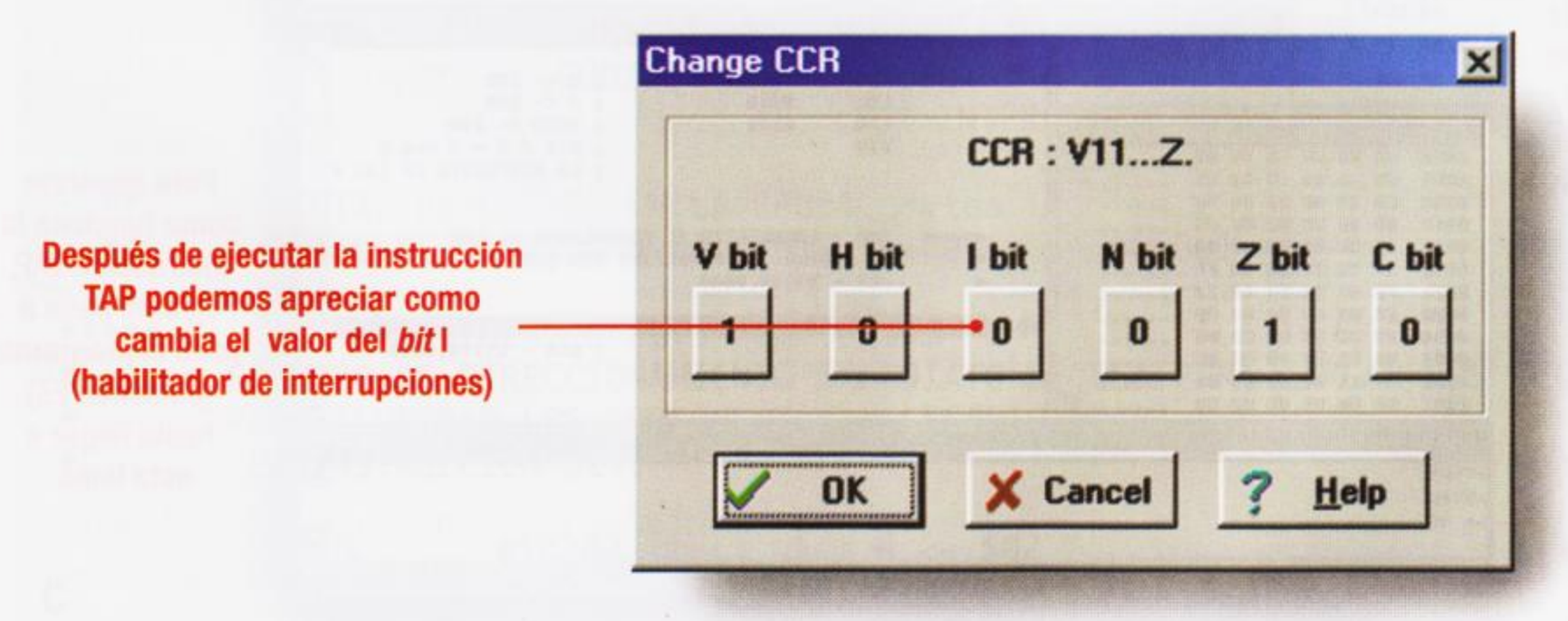
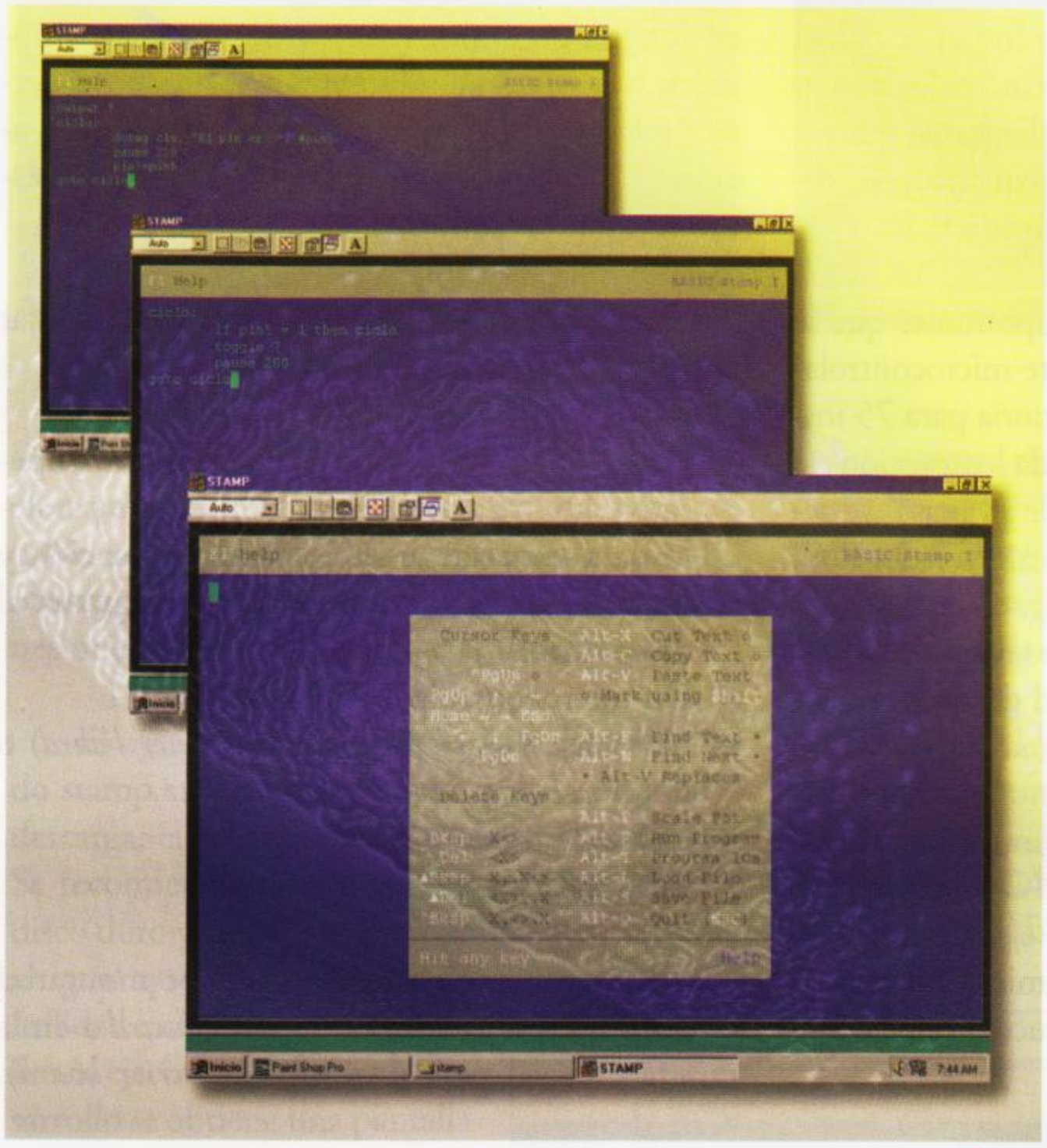


Figura 8.35 *Después de ejecutar la línea con la instrucción TAP, si procedemos a visualizar de nuevo el registro CCR, podremos ver que han cambiado los valores de sus bits. En este registro casi todos los bits son solo de lectura, por lo que solo cambian por las condiciones de la CPU. El bit I sí se puede escribir, el cual corresponde al habilitador gobal de las interrupciones.*

LECCIÓN 9



EJERCICIOS DE PROGRAMACIÓN CON EL BASIC Stamp

Ahora centraremos nuestra atención en la programación de los famosísimos microcontroladores BASIC Stamp I y II, incluyendo los pasos involucrados en el desarrollo de soluciones inteligentes con estos poderosos dispositivos. Podremos captar la facilidad de trabajo y la gran cantidad de herramientas que ellos nos ofrecen para reducir significativamente el desarrollo de productos electrónicos.

Como es lógico, antes de empezar a desarrollar un programa para un microcontrolador se debe conocer bien su arquitectura, sus instrucciones, su mapa de memoria, y en general, todo lo relacionado con su *hardware*. El BASIC Stamp no es una excepción, por lo tanto, con el fin de recordar los elementos principales que lo componen, se ha elaborado un diagrama que representa los elementos que lo constituyen, dicho esquema se muestra en la **figura 9.1**.

Uno de los aspectos más importantes que se deben tener en cuenta es que este microcontrolador sólo tiene capacidad de memoria para 75 instrucciones. Ello se debe a que cada instrucción escrita en lenguaje PBASIC equivale a varias instrucciones de ensamblador del microcontrolador PIC, incluido en el módulo del Stamp, el cual está encargado de interpretar los códigos recibidos. Otro factor importante es la velocidad de ejecución de las instrucciones, limitadas a un máximo de 2.000 por segundo. La razón fundamental para ello es la misma que se expuso anteriormente para la capacidad de memoria, mientras el PIC posee un oscilador de 4 MHz que le permite ejecutar una instrucción de ensamblador cada microsegundo, el tiempo de ejecución de una instrucción en lenguaje PBASIC es mucho mayor.

Desarrollo de programas

Quizás la principal ventaja de los BASIC Stamp es la facilidad que ofrecen para el desarrollo de programas en muy corto tiempo, con los beneficios adicionales que entrega la programación en lenguaje de alto nivel. Además, estos módulos se programan directamente desde el puerto de la computadora, evitando así la utilización de programadores especiales o costosos sistemas de desarrollo.

Los elementos que se requieren para trabajar con el BASIC Stamp I son:

- Una computadora tipo PC con sistema operativo DOS 2.0 ó superior, con puerto paralelo para conectar el cable de programación y un *drive* de 3,5" para copiar el programa al disco duro.
- Un microcontrolador BASIC Stamp I
- Una tarjeta para prototipos (*carrier board*) o un *protoboard*
- Cable de programación
- Programa stamp.exe

Para empezar a trabajar, se debe montar el circuito o experimento en un *protoboard* o en la tarjeta especial para prototipos (*carrier board*).

Figura 9.1 *Representación del microcontrolador BASIC Stamp I*

Figura 9.2 Aspecto del programa stamp.exe

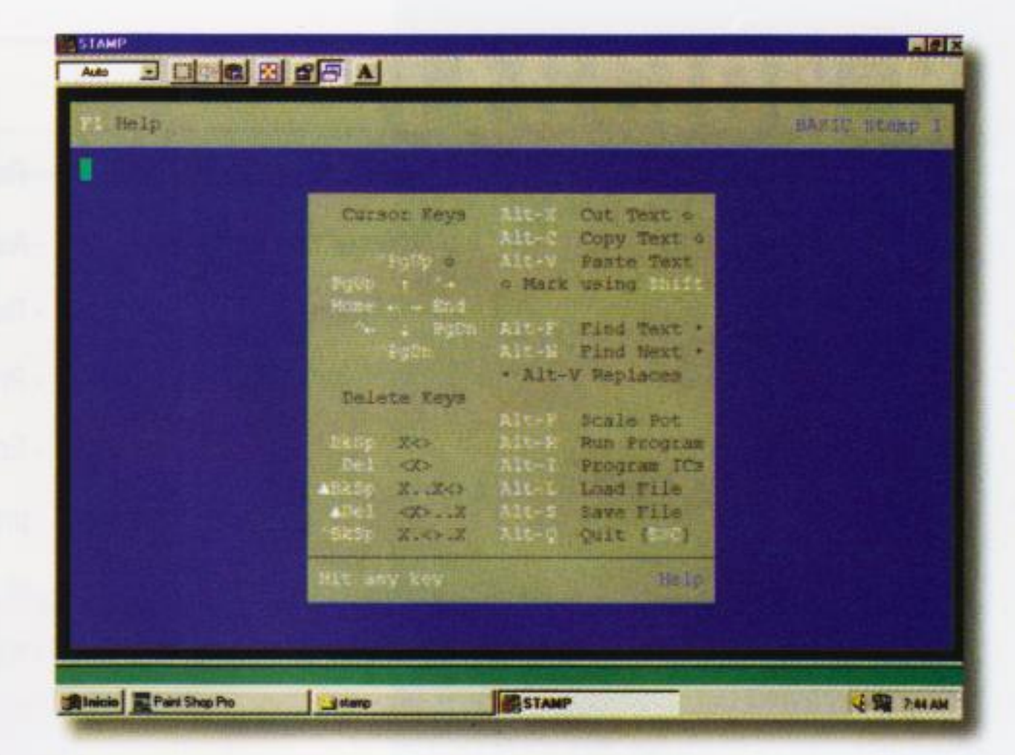


Figura 9.3 Menú de ayuda en pantalla (F1)

Después, se debe conectar el cable de programación entre el puerto paralelo y el circuito. Este cable se puede adquirir junto con el BASIC Stamp o se puede construir según las conexiones que se muestran en los circuitos.

El siguiente paso es instalar el *software* llamado stamp.exe, el cual permite editar, compilar y descargar el programa hacia el microcontrolador. Se recomienda crear un directorio nuevo en el disco duro y copiar allí el programa mencionado, al igual que los programas de utilidad que se vayan elaborando. Al ejecutar el *software* de desarrollo se obtiene una pantalla como la que se muestra en la **figura 9.2**, que, como se puede apreciar, es bastante simple. Esto es una ventaja ya que se puede trabajar en computadoras nuevas y en máquinas viejas, que estaban destinadas a salir de circulación.

Hecho lo anterior, sólo resta escribir el programa propio de la tarea y mediante algunos comandos especiales, compilarlo y descargarlo hacia el microcontrolador, a través del cable de programación. Por facilidad, los comandos especiales con que se debe trabajar, se pueden observar en pantalla con sólo pulsar la tecla F1. En la **figura 9.3** se muestra el menú de ayuda que se despliega en la pantalla, y en la **figura 9.4** una explicación más detallada de los mismos.

Durante el proceso de compilación o en el momento de grabar el microcontrolador, se podría presentar algún problema, ya sea de sintaxis o de comunicación entre los dispositivos. Por tal motivo, es posible que en algún momento apa-

Tecla	Función	Comentario
F1	Presenta el menú de ayuda	
ALT-R	**Run**: descarga el programa hacia el STAMP.	Muestra en forma gráfica la descarga del programa e indica la porción de memoria utilizada
ALT - L	**Load**: carga un programa ya existente	
ALT -S	**Save**: Graba el programa	
ALT -I:	**Info**: Muestra la versión del chip intérprete	Sólo en el BASIC Stamp II
ALT -M	**Memory**: Muestra un mapa de la memoria utilizada	Sólo en el BASIC Stamp II
ALT -Q	**Quit**: Termina el programa	
ALT -P	**Pot**: Determina el valor de la escala usada en la instrucción Pot	Sólo en el BASIC Stamp I
ESC	Termina el programa	

Figura 9.4 Comandos utilizados en el desarrollo de programas

Mensaje	Causa	Posibles soluciones
Hardware not found	El programa no pudo establecer comunicación con el Stamp	- Revise la fuente de alimentación del stamp - Asegúrese que el programa esté funcionando bajo DOS - Revise el cable de conexión - Pruebe con un puerto diferente - En el caso del Stamp II, salga del programa y vuelva a entrar, especificando el puerto que utiliza (Stamp 2/2 ó Stamp 2/1)
EEPROM verify failed	Un problema ocurrió cuando el software fue leído o descargado del BASIC Stamp	- Revise la fuente de alimentación - Asegúrese de trabajar en DOS - Intente reconectar el cable de programación
Varios mensajes de error y parte del texto se ilumina	Existe un error de sintaxis en el programa que se escribió	- Revise la sintaxis del programa y vuelva a intentarlo

Figura 9.5 *Mensajes de error que se pueden presentar*

rezca un mensaje de error en la pantalla, razón por la que consideramos de gran utilidad el resumen que se muestra en la **figura 9.5** y que incluye los mensajes de error más comunes, sus posibles causas y la forma de solucionarlos.

El orden lógico en que se deben seguir los pasos del desarrollo de un programa, es el siguiente:

- Se ejecuta el *software* stamp.exe
- Se escribe en la pantalla el programa correspondiente
- Se conecta el cable de programación entre la computadora y el circuito
- Se enciende la fuente de alimentación del circuito
- Se compila y se descarga el programa hacia el microcontrolador, con el comando ALT-R
- Antes de salir del programa de desarrollo se debe guardar el programa, o los cambios que se hayan hecho

Como sucede con casi todo, la práctica es la que nos genera un verdadero conocimiento. Por tal razón, veremos a continuación unos ejercicios simples de programación con los cuales esperamos se disipen las dudas que puedan existir.

Experimento No. 13
Encender y apagar un diodo LED

Este es el experimento básico por excelencia y quizás uno de los que brinda mayor satisfacción, dado que representa el ingreso a este maravilloso mundo de los microcontroladores. En la **figura 9.6** se muestra el diagrama del circuito, incluyendo el cable de programación. Para este proyecto sólo se requiere una fuente de alimentación externa de 5VCC.

En este punto, llegamos a las instrucciones o comandos que se deben escribir, de acuerdo a la tarea específica que se va a cumplir. En el artículo sobre los BASIC Stamp que se publicó en la edición anterior, se entregó la lista completa de las instrucciones propias de este módulo.

A partir de ahora, y en todos los ejercicios que realicemos, iremos explicando la esencia de cada instrucción y los parámetros que involucran. De todas formas, quienes están familiarizados con el lenguaje BASIC tradicional, reconocerán sin duda alguna la estrecha relación que éste guarda con el nuevo lenguaje en que trabajamos, llamado PBASIC.

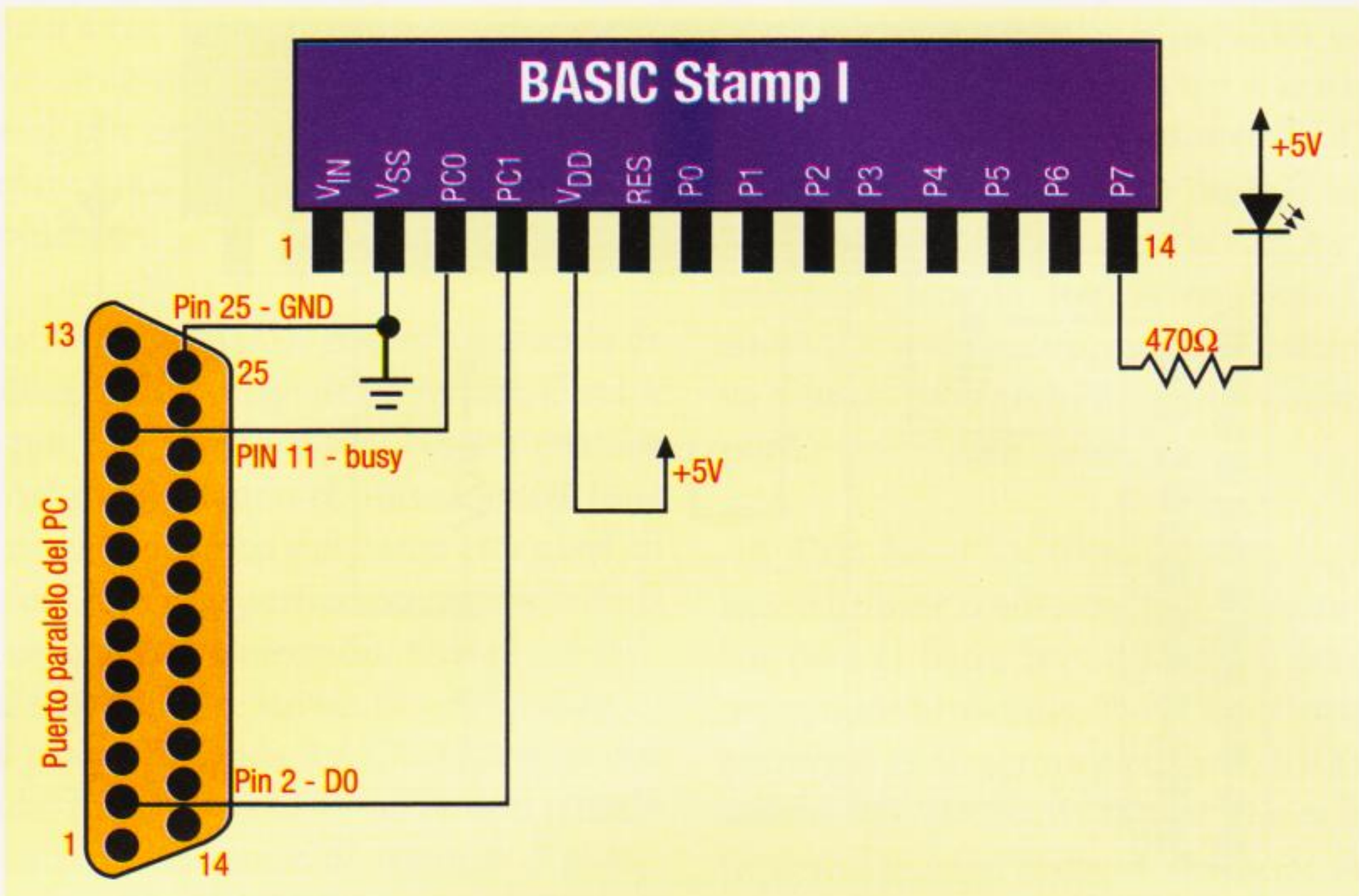


Figura 9.6 *Circuito utilizado para el primer experimento. Encender y apagar un diodo LED*

Primer caso. En la **figura 9.7** se muestra el primer programa que se escribió para encender y apagar el diodo LED conectado al pin de entrada/salida número 7 del Stamp. En este caso no se requieren demasiados comentarios ya que es muy lógico e intuitivo, un claro ejemplo de las bondades de la programación en un lenguaje de alto nivel. Las instrucciones *High* (alto) y *Low* (bajo), configuran el pin correspondiente como salida, a la vez que le dan el nivel lógico indicado por su propio nombre. El parámetro utilizado por ellos es el número del pin referido (en este caso el pin 7).

La instrucción PAUSE (retardo) genera un tiempo de espera, su parámetro es el número que se escribe frente a ella y que está dado en milisegundos. Usted puede hacer diferentes pruebas cambiando el número de milisegundos para observar el resultado. En cuanto a la instrucción GOTO CICLO, ésta sólo hace que el programa se siga ejecutando indefinidamente.

Segundo caso. En la **figura 9.8** se muestra el segundo programa escrito para encender y apagar el diodo LED. En este caso, utilizamos una instrucción del Stamp llamada TOGGLE, como su tra-

Figura 9.7 *Programa LED1.BAS. Enciende y apaga un diodo LED*

Figura 9.8 *Programa LED2.BAS. Enciende y apaga un diodo LED*

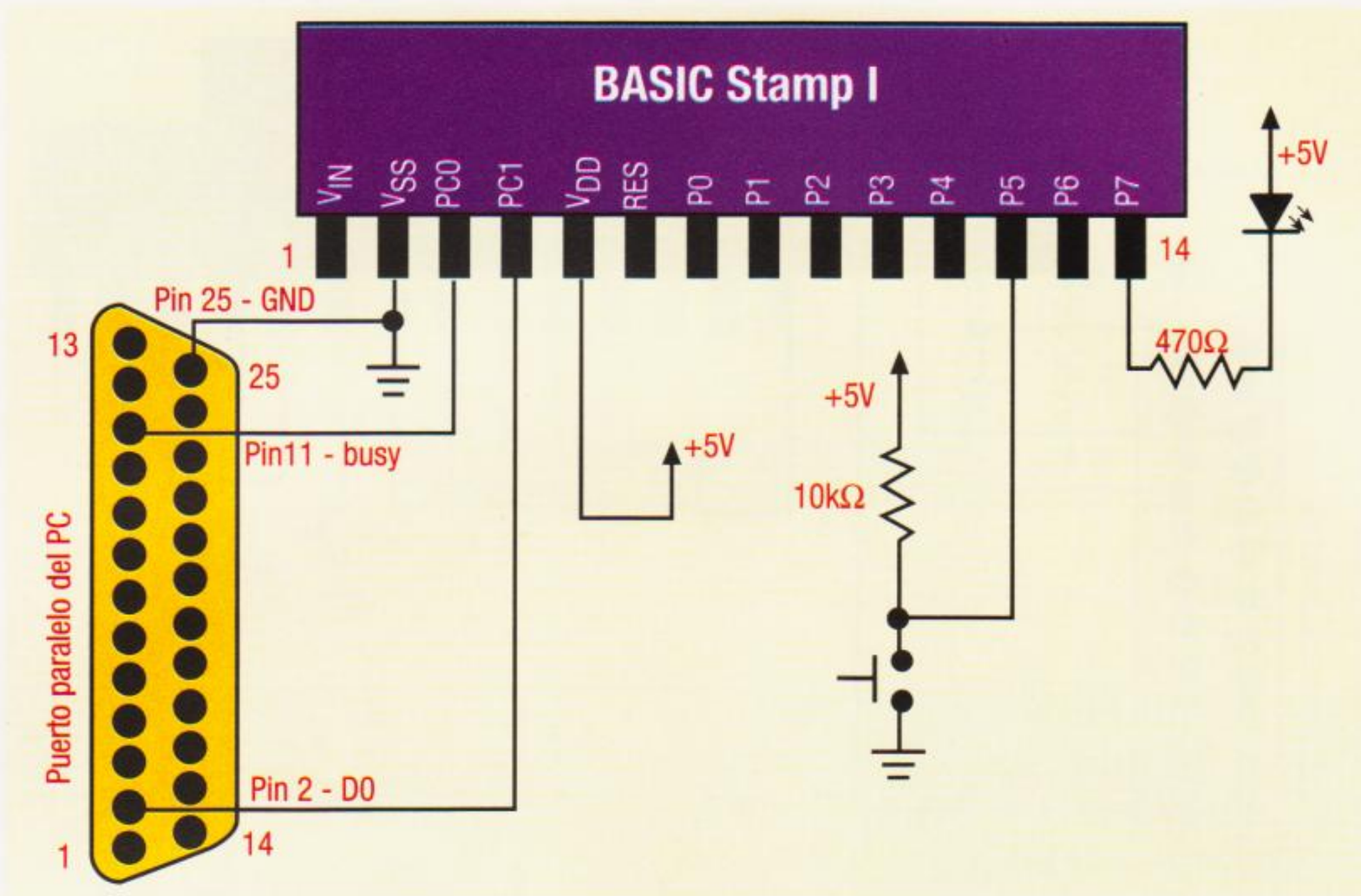


Figura 9.9 *Circuito de prueba con un LED y un botón pulsador*

ducción lo indica, sirve para conmutar el estado de un pin. En este caso el pin 7, que es donde está conectado el LED. Luego de la instrucción que cambia el estado lógico del pin, se agrega un retardo generado por la instrucción PAUSE. Si este retardo no se pone en este sitio, el LED se vería permanentemente encendido ya que la conmutación sería demasiado rápida. Nuevamente, se utiliza la etiqueta CICLO y la instrucción GOTO CICLO para ejecutar el programa indefinidamente.

Experimento No. 14
Controlando el diodo LED con un pulsador

En la **figura 9.9** se muestra el circuito utilizado para el experimento, allí se ha adicionado un botón pulsador conectado al pin 5 del Stamp. En ese punto también se ha puesto una resistencia que sirve para fijar un nivel lógico alto cuando éste no es pulsado. La idea es leer el estado del pulsador y de acuerdo a éste (pulsado o no pulsado), tomar la decisión de encender o dejar apagado el LED. Este ejercicio es muy importante ya que por primera vez se hace lectura de señales externas al microcontrolador.

Primer caso. En la **figura 9.10** se muestra el primer programa que se escribió para este experimento. Allí tenemos un ciclo en el que se verifica el estado del pin número 5, donde está conectado el pulsador; luego se hace un retardo de algunos milisegundos, y, por último, se asigna el valor leído en la entrada al pin de salida número 7, donde está conectado el LED.

La instrucción DEBUG es muy especial, sirve para mostrar en la pantalla de la computadora el valor o estado de algunas variables presentes en el microcontrolador. Por lo tanto, se utiliza frecuentemente durante el desarrollo de algún programa para monitorear como se está eje-

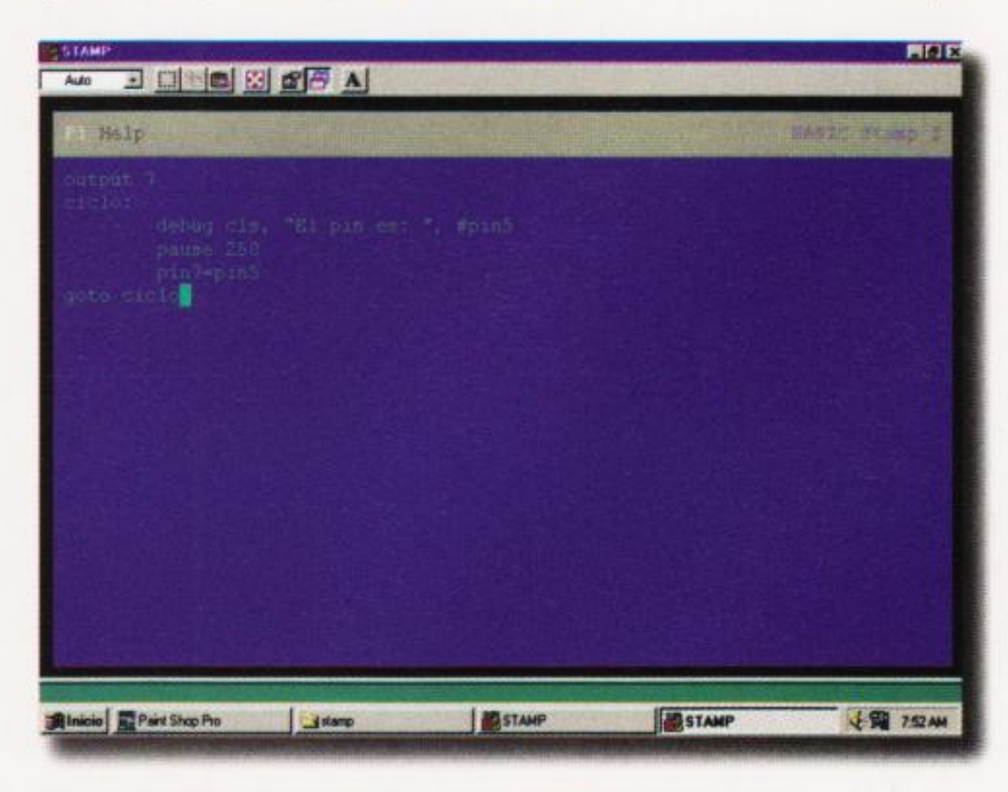


Figura 9.10 *Programa LED3.BAS. Controla un diodo LED*

cutando una tarea. En este caso particular se utiliza para mostrar en la pantalla el estado lógico que tiene el pin número 5, así podemos ver el cambio entre unos y ceros cada vez que es oprimido el pulsador.

La instrucción PAUSE genera un retardo de varios milisegundos, luego de los cuales, el valor leído del pin 5 es pasado al pin 7 para controlar el LED. Así pues, cuando el pulsador está libre se lee un nivel lógico alto y al poner este nivel en la salida del LED, éste permanece apagado dada la forma en que está conectado. Cuando el pulsador está oprimido, se lee un nivel lógico bajo, el cual, al pasar a la salida del LED permite que se encienda. En algunos casos se nota el retardo que hay entre el momento de oprimir el pulsador y el encendido del LED, esto se debe a la instrucción PAUSE.

Segundo caso. En la **figura 9.11** se muestra el segundo programa correspondiente a este ejemplo, como se puede ver, es más simple que el anterior. En este caso, recurrimos a una de las herramientas de programación más conocidas en los lenguajes de alto nivel, los ciclos condicionales. En la línea if pin5 = 1 then ciclo, se prueba el estado lógico que posee el pin donde se conectó el pulsador. Si éste no está oprimido, el nivel lógico es 1 y por lo tanto el programa regresa a la etiqueta marcada ciclo y vuelve a hacer la pregunta.

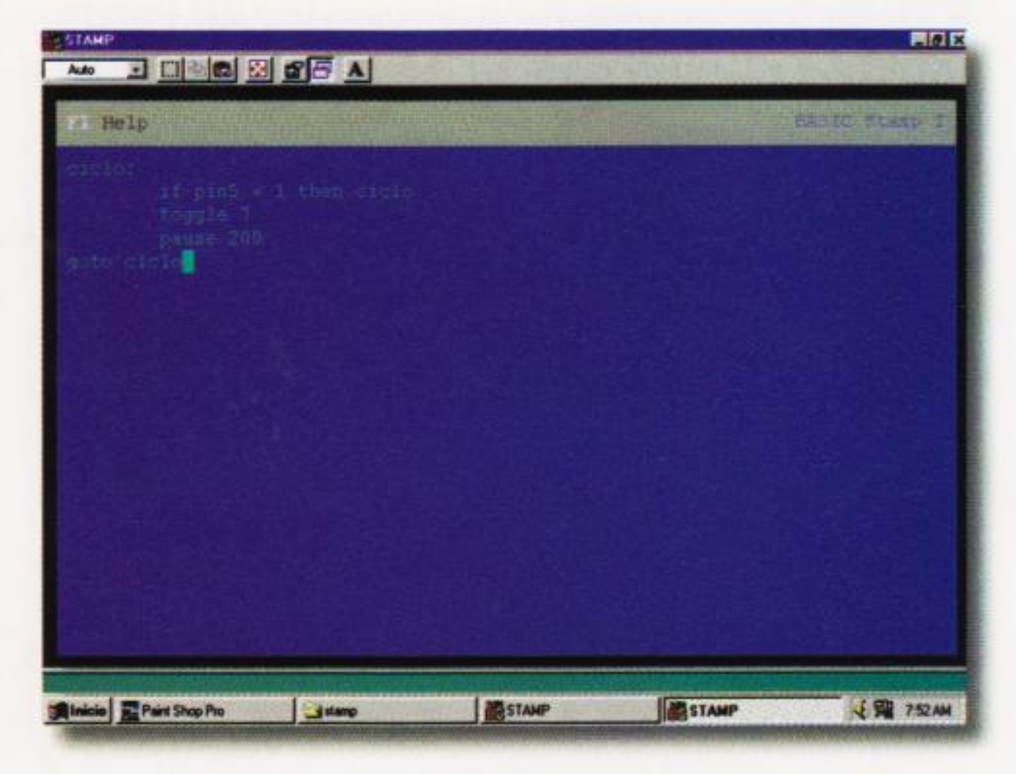


Figura 9.11 *Programa LED4.BAS. Controla un diodo LED*

Si por el contrario, el pulsador está oprimido, se leerá un nivel lógico 0 y el microcontrolador ejecutará las instrucciones TOGGLE 7 y PAUSE, lo que genera un cambio en el estado lógico del pin que maneja el LED y genera un tiempo de retardo, respectivamente. Esto quiere decir que, si se deja oprimido el pulsador, el LED va a estar cambiando de estado constantemente por lo que se verá titilar.

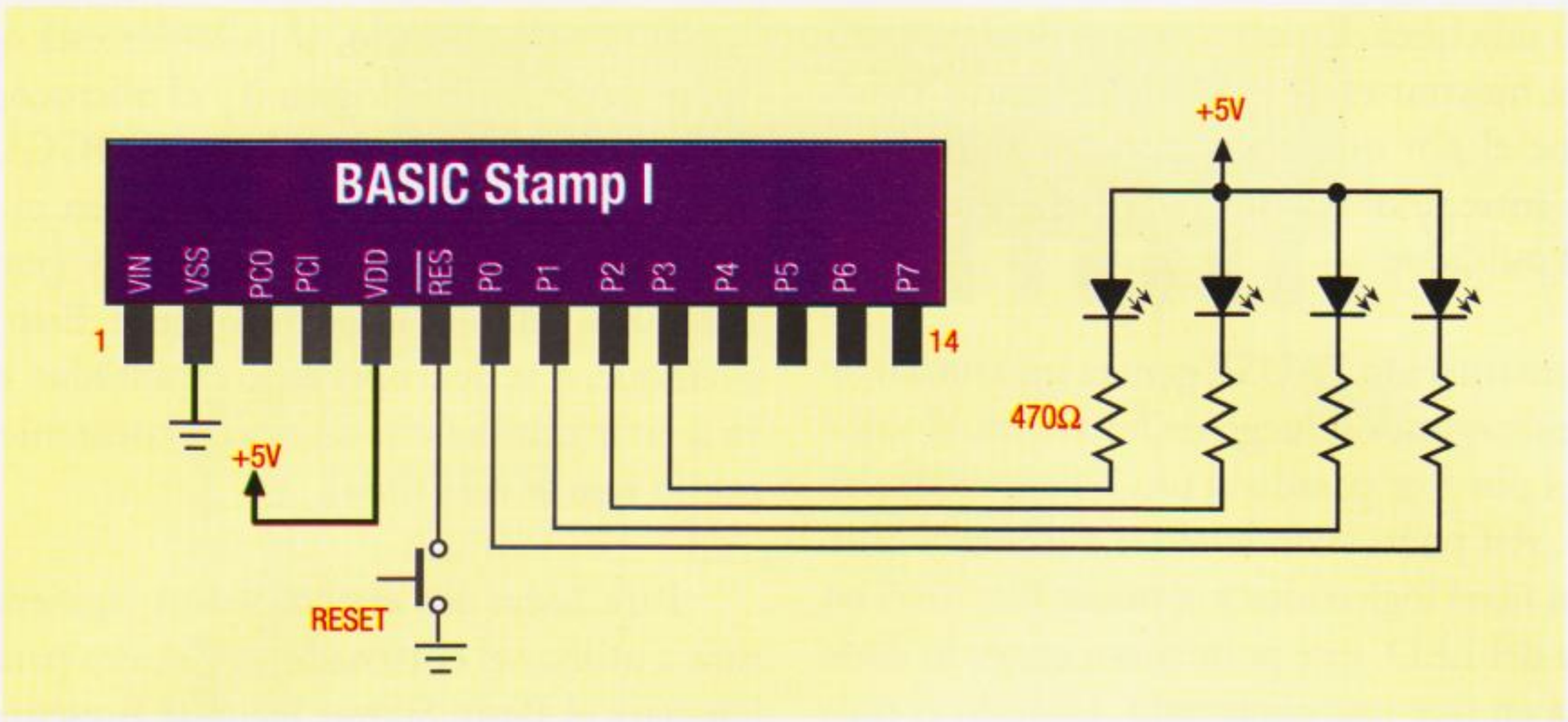
Para hacer los experimentos siguientes vamos a utilizar el entrenador o circuito para pruebas para el Basic Stamp I que se presentó en la sección de proyectos. Para cada experimento se entregará el diagrama pictórico en donde se presentan las conexiones que se deben hacer en el entrenador para hacer el ejercicio, el diagrama esquemático y, por último, el programa que se debe grabar en el microcontrolador.

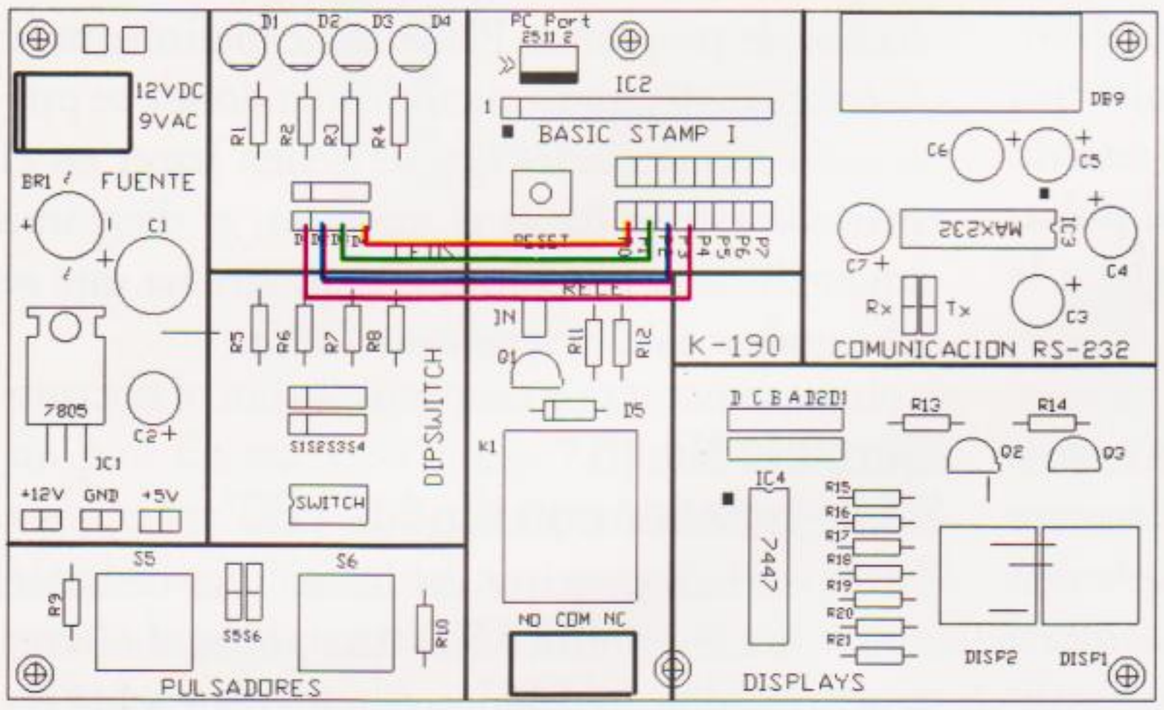
Ejercicio No.15
Secuenciador con diodos LED

Este ejercicio consiste en hacer una rotación en los cuatro diodos LED que posee el circuito entrenador, los cuales se conectan a los pines de entrada/salida P0 a P3 del microcontrolador. En la **figura 9.12** se muestran los diagramas esquemático y pictórico del experimento. El programa que se escribió en el microcontrolador incluye varias instrucciones nuevas para nosotros, a continuación explicaremos cada una de ellas.

La instrucción LET DIRS permite configurar los puertos del microcontrolador; en este caso, un uno (1) configura el pin correspondiente como salida y un cero (0) lo hace como entrada. El valor con que se programa la dirección de los puertos está precedido por un signo de porcentaje (%) lo que significa que está dado en binario.

El ciclo principal del programa consiste en llevar a los pines del puerto el valor correspondiente a las rotaciones, con el fin de lograr un efecto visual agradable.






```
let dirs = %00001111
rotar:
     for b2 = 0 to 3
      LOOKUP b2, (1,2,4,8),b3
      let pins = b3
      pause 1000
     next
goto rotar
```

Figura 9.12 *Secuenciador con diodos LED*

El ciclo FOR realiza el mismo proceso durante cuatro veces, dicha cuenta se lleva en la variable o posición de memoria RAM llamada b2. La instrucción LOOKUP se encarga de pasar a la variable b3 los números que están indicados dentro del paréntesis.

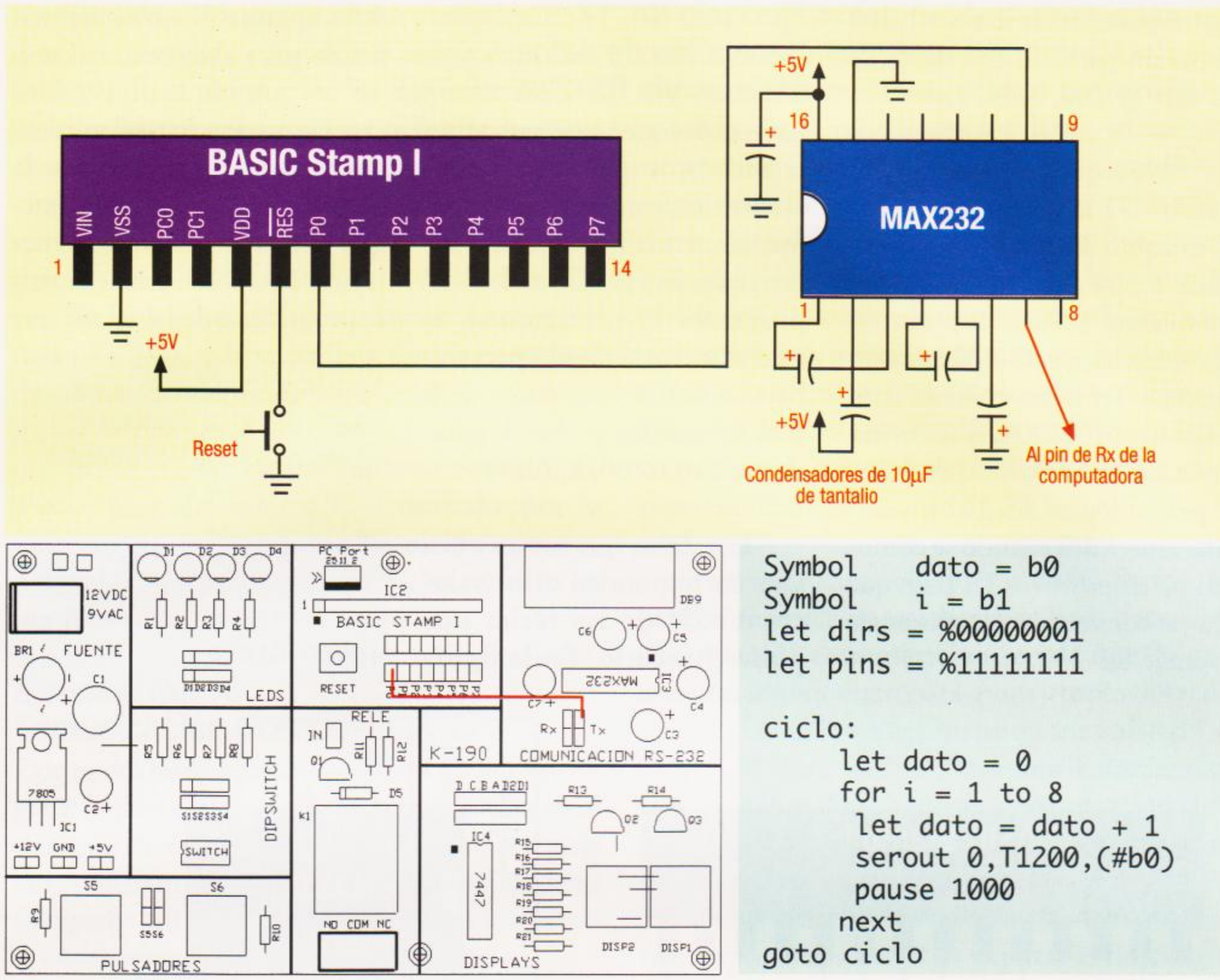
La instrucción **Let pins** pasa el contenido de la variable b3 hacia los pines del microcontrolador. De esta forma, los números 1, 2, 4 y 8 se llevan a los puertos en forma secuencial, formando una rotación en los LED del entrenador. La instrucción **Pause** 1000 genera un retardo de 1 segundo (1.000 ms) entre cada rotación.

El uso de la instrucción LOOKUP es bastante interesante ya que permite hacer una secuencia de números, sin que tenga que existir alguna lógica u orden establecido entre ellos.

Ejercicio No. 16
Transmisión de datos seriales vía RS-232

Este experimento consiste en transmitir una serie de datos desde el microcontrolador Basic Stamp I hacia una computadora tipo PC, en la cual los datos se reciben por un puerto serial y se muestran en la pantalla. En nuestro caso, recurrimos al uso del programa Hyperterminal de Windows 95 para la recepción de los datos ya que es de fácil uso y se encuentra disponible en casi cualquier computadora.

Para convertir los datos que entrega el microcontrolador en lógica TTL a niveles RS-232, se utiliza un circuito integrado referencia MAX232 y un conjunto de condensadores, tal como se muestra en la **figura 9.13**. El pin de transmisión de este sistema debe conectarse al pin de recepción de la computadora. Para ello, el entrenador posee un cable serial especialmente construido para estos experimentos.




```
Symbol    dato = b0
Symbol    i = b1
let dirs = %00000001
let pins = %11111111

ciclo:
     let dato = 0
     for i = 1 to 8
      let dato = dato + 1
      serout 0,T1200,(#b0)
      pause 1000
     next
goto ciclo
```

Figura 9.13 *Transmisión de datos seriales vía RS-232*

Para convertir los datos que entrega el microcontrolador en lógica TTL a niveles RS-232, se utiliza un circuito integrado referencia MAX232 y un conjunto de condensadores, tal como se muestra en la **figura 9.13**. El pin de transmisión de este sistema debe conectarse al pin de recepción de la computadora. Para ello, el entrenador posee un cable serial especialmente construido para estos experimentos.

El programa empieza por la definición de dos variables, *dato* e *i*, las cuales ocuparán las posiciones de memoria RAM llamadas b0 y b1. Luego, se hace la programación de los puertos con la instrucción LET DIRS, dejando como salida únicamente el pin P0 del microcontrolador. Además, el pin P0 se inicializa en un nivel lógico alto con la instrucción de cargar el puerto llamada LET PINS.

El ciclo principal del programa consiste en hacer un simple contador de 1 a 8 y enviar por la salida serial dicho número de tal forma que, en la pantalla de la computadora, vayan apareciendo en forma secuencial. Para ello, recurrimos a un ciclo FOR que se repite 8 veces, utilizando la variable *dato* para hacer la cuenta. La instrucción SEROUT se encarga de sacar por un pin el dato en formato RS-232.

Dicha instrucción tiene varios parámetros de configuración que son: el pin del puerto por donde se sacan los datos serialmente (en este caso el pin 0), la velocidad de transmisión (1.200 bps) y el dato o valor que se va a transmitir (corresponde a la variable b0 ó *dato*). La instrucción PAUSE genera un retardo de 1 segundo entre el envío de cada dato serial. Obviamente, en la computadora el programa Hyperterminal se confi-

guró para recibir datos a 1.200 bps, sin paridad, con datos de 8 *bits* (1.200, 8, N, 1).

Note que en la instrucción SEROUT la velocidad se escribió como T1200 lo que significa 1.200 bps con polaridad verdadera (*True*); ésta corresponde a lógica RS-232 convencional. También existe la configuración N1200, que corresponde a una velocidad de 1.200 bps con lógica RS-232 invertida; ésta es útil cuando se comunican dispositivos TTL sin que haya conversión a valores de voltaje RS-232.

Ejercicio No. 17 Recepción de datos seriales vía RS-232

Este experimento consiste en recibir datos que llegan serialmente desde una computadora mediante una interfaz RS-232. Los datos son enviados desde el programa Hyperterminal, el cual se ha configurado para trabajar a 1.200 bps, datos de 8 *bits*, sin paridad y 1 *bit* de parada o *stop* (1.200, 8, N, 1).

Los datos que se van a enviar corresponden a los valores ASCII de las teclas que pulse el usuario. En la **figura 9.14** aparece el *hardware* utilizado para el ejercicio.

El programa del microcontrolador se inicia con la definición de la variable *dato*, que ocupa la posición de memoria RAM b0. Los puertos se programan de tal forma que

```
Symbol    dato = b0
let dirs = %00001111
let pins = %00000000

ciclo:
     serin 7,T1200,b0
     let pins = dato

goto ciclo
```

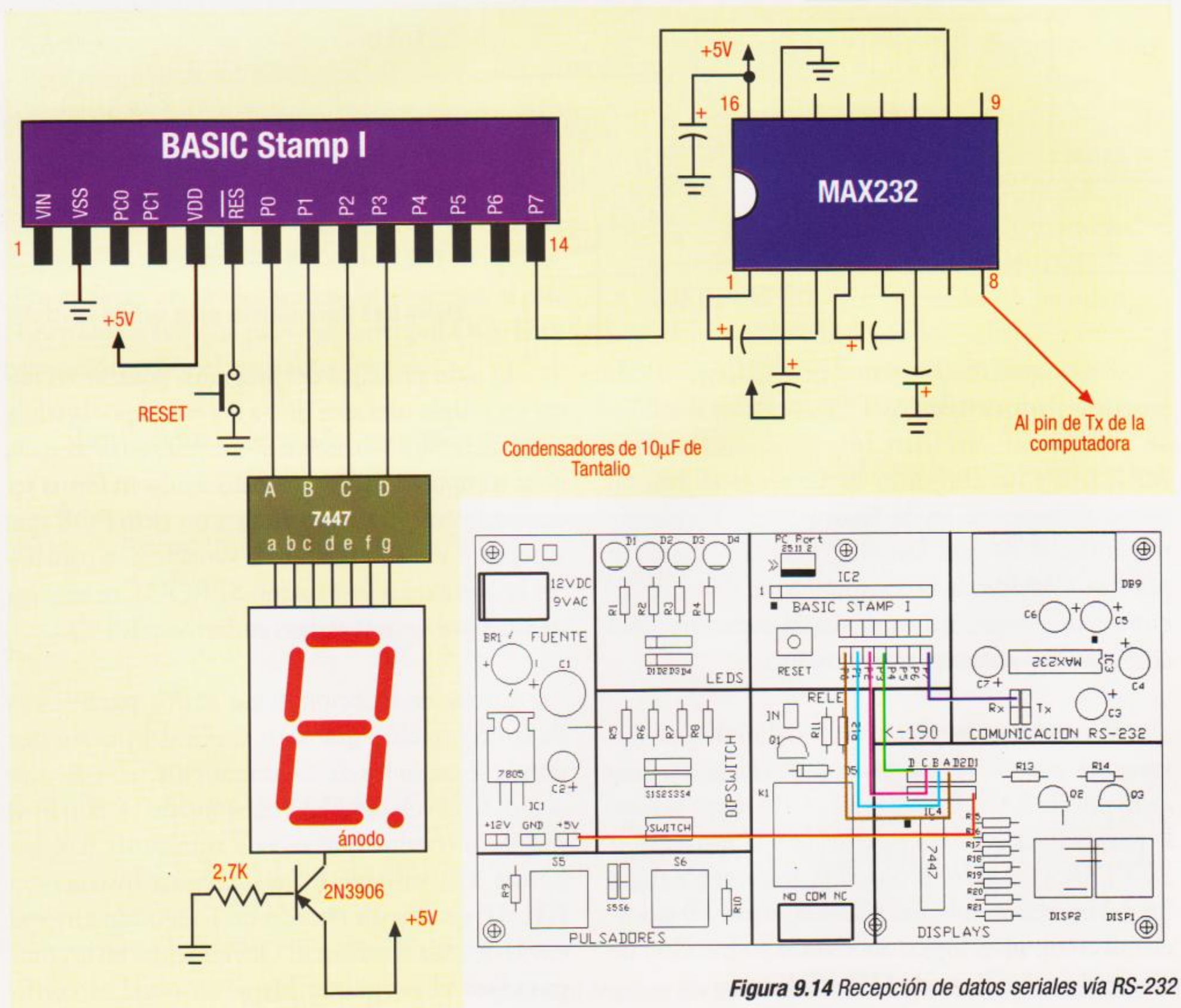


Figura 9.14 Recepción de datos seriales vía RS-232

los pines P0 a P3, que manejan el *display*, queden como salidas y los demás como entradas, incluyendo el pin que lee los datos seriales. El ciclo principal del programa se limita a recibir datos con la instrucción SERIN y luego los pasa al *display*. Note que la instrucción SERIN tiene varios parámetros que deben ser especificados: el pin por donde se reciben los datos (el pin 7), la velocidad de los mismos y su polaridad (T1200), y la variable a donde se pasa el dato que es recibido (posición b0).

En el *display* sólo se muestra el dígito de menor peso del valor ASCII recibido, por lo tanto, si se oprimen las teclas de los números del 0 al 9, se puede apreciar dicho número en los *displays*.

Ejercicio No. 18
Comunicación serial sincrónica

Existen dos formas de intercambiar información binaria: la paralela y la serial. La comunicación serial transmite un *bit* a la vez, por lo cual posee la ventaja de necesitar un menor número de líneas para la transferencia de la información y las distancias a las cuales se puede hacer el intercambio son mayores.

Existen dos formas de comunicación serial: la sincrónica y la no sincrónica. En la comunicación sincrónica, además de una línea sobre la que se transfieren los datos, se necesita otra que contenga pulsos de reloj que indiquen cuando cada *bit* de datos es válido; la duración del *bit* está determinada por la duración del pulso de sincronismo. En la comunicación no sincrónica los pulsos de reloj no son necesarios y se acude a otros mecanismos para hacer la lectura/escritura de los datos; la duración de cada *bit* sobre la línea está determinada por la velocidad con la cual se hace la transferencia de datos.

Los microcontroladores BASIC Stamp II poseen instrucciones especiales para hacer los dos tipos de comunicación serial. Para la comunicación no sincrónica, por ejemplo del tipo RS-232, se utilizan las instrucciones SERIN y SEROUT para recibir y transmitir datos, respectivamente. Para hacer la comunicación sincrónica, que es la que nos interesa, existen las instrucciones denominadas SHIFTIN y SHIFTOUT, las cuales permiten recibir y enviar datos, respectivamente. Para explicar el funciona-

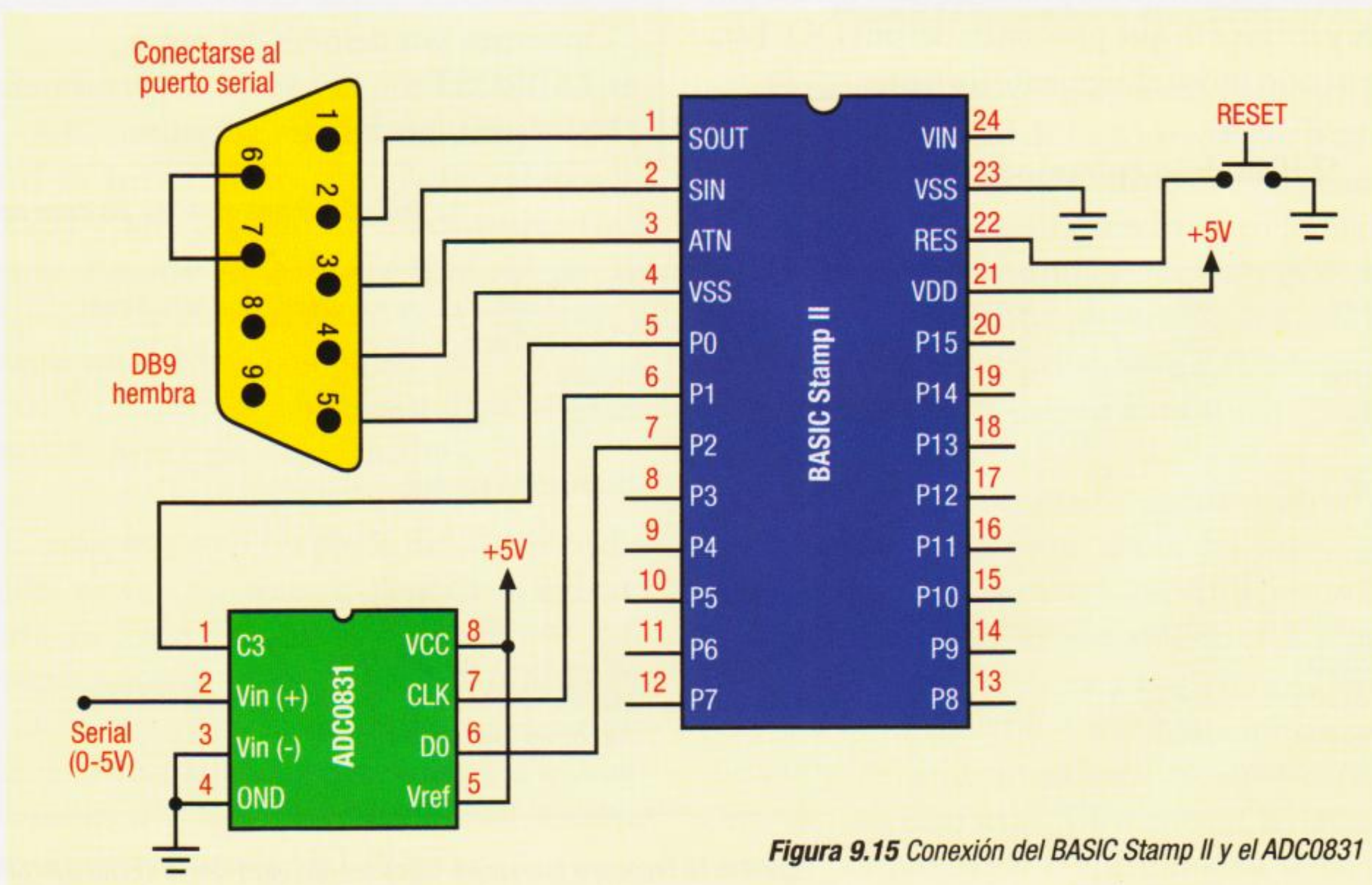


Figura 9.15 *Conexión del BASIC Stamp II y el ADC0831*

miento de estas instrucciones vamos a leer un convertidor A/D serial de 8 *bits*, referencia ADC0831 de National Semiconductor.

Lectura del convertidor ADC0831. En la **figura 9.15** se muestra el diagrama esquemático del circuito construido. El convertidor ADC0831 puede leer una señal análoga con un intervalo de variación entre 0 y 5V, y, entrega en su salida, de forma serial, el equivalente digital de 8 *bits* de la señal leída. Este convertidor posee tres pines de control para hacer la comunicación: el pin llamado CS (*chip select*) se debe habilitar antes de hacer una lectura (pasar a nivel lógico bajo) y se debe deshabilitar al terminarla (nivel lógico alto), el pin CLK (reloj) debe recibir una onda cuadrada en la cual cada pulso habilita la salida de un *bit* de datos a través del pin DO.

El programa escrito para que el microcontrolador lea el dato del convertidor y lo presente en la pantalla de la computadora, se muestra en la **figura 9.16**. En dicho listado se utiliza la instrucción SHIFTIN para leer el dato serial; este comando se encarga de controlar los dos pines de comunicación, es decir, genera los pulsos de reloj para el pin CLK y lee los *bits* que provienen del pin DO. Esta instrucción posee el siguiente formato:

Shiftin dpin,cpin,mode,[result\bits]

- Dpin es una variable o constante (entre 0 y 15) que especifica el pin que se conecta a la línea de datos.
- Cpin es una variable o constante (entre 0 y 15) que especifica el pin en el cual se conectará la señal de reloj para la comunicación serial.
- Mode es un valor entre 0 y 3 que especifica en que forma se recibirán los datos seriales, trabaja como se indica en la **tabla 9.1**.
- Result es un *bit*, *nibble*, *byte* o palabra en la cual se almacenará el dato recibido.
- bits es un parámetro opcional en el cual se especifica cuantos *bits* de datos se recibirán (1 a 16), por defecto está configurado para datos de 8 *bits*.

La instrucción DEBUG se encarga de mostrar en la pantalla de la computadora el dato recibido del convertidor. Obviamente, el cable de programación debe permanecer conectado a la PC para que funcione correctamente. El resto del programa es muy sencillo y puede ser comprendido fácilmente.

- **MSBPRE 0** Lee el *bit* MSB primero, muestrea *bits* antes del pulso de reloj
- **LSBPRE 1** Lee el *bit* LSB primero, muestrea *bits* antes del pulso de reloj
- **MSBPOST 2** Lee el *bit* MSB primero, muestrea *bits* después del pulso
- **LSBPOST 3** Lee el *bit* LSB primero, muestrea *bits* después del pulso

Tabla 9.1 *Forma de recibir los datos seriales*

```
ADres       var        byte                    ' resultado de la conversión: un byte
CS          con        0                       ' Chip select
AData       con        1                       ' Datos
CLK         con        2                       ' Reloj

high        CS                                 ' Deshabilita ADC

again:
  low       CS                                 ' Activa el ADC0831.
  shiftin   AData,CLK,msbpost,[ADres\9]        ' Lee los datos
  high      CS                                 ' Desactiva el ADC0831
  debug ?   ADres                              ' Muestra el resultado en la pantalla
  pause     1000                               ' Retardo de un segundo
goto  again                                    ' Vuelve a empezar
```

Figura 9.16 *Programa que lee los datos del ADC0831 con la instrucción SHIFTIN*

LECCIÓN 10

PROGRAMACIÓN DE MICROCONTROLADORES INTEL

En la sección de teoría se analizan aspectos como la arquitectura, la organización de la memoria y los módulos periféricos. Basándonos en esa información, podemos analizar brevemente algunos aspectos relacionados con su modo de programación y desarrollaremos algunos ejercicios básicos que nos darán una base para introducirnos en esta tecnología.

Los microcontroladores Intel siguen siendo considerados entre los más poderosos del mercado, pues poseen un considerable número de instrucciones y modos de direccionamiento para acceder a la memoria RAM. En esta introducción a esta tecnología de microcontroladores estudiemos sus modos de direccionamiento y su conjunto de instrucciones agrupados por categorías así:

- Instrucciones aritméticas
- Instrucciones lógicas
- Instrucciones para la transferencia de datos
- Instrucciones booleanas
- Instrucciones de salto

Modos de direccionamiento

- Direccionamiento directo

En este caso, el operando se especifica dentro de la misma instrucción. Ejemplo:

ADD A,2BH. Esta instrucción significa sumar (ADD) el contenido del acumulador con el contenido de la posición de memoria (2B). A ← (A) + (2B)

Recuerde que un número o registro entre paréntesis se refiere al contenido, así:

(A) es el contenido del acumulador
(2B) es el contenido de la posición de memoria 2B

- Direccionamiento indirecto

La instrucción especifica un registro que contiene la dirección del operando. Los registros para direccionar sobre la memoria RAM son el R0 y el R1 del banco de registros seleccionados, o el puntero a la pila (*stack pointer*). El carácter (@) es el identificador de direccionamiento indirecto. Ejemplo:

ADD A,@R0. Así, si (R0) = 4BH la operación será: A ← (A) + (2B).

- Direccionamiento por registro

Los microcontroladores 8051continen cuatro bancos de registros seleccionados por los *bits* 3 y 4 del PSW. Cada banco de registros posee ocho registros (R0...R7). En este tipo de direccionamiento el propio código de la instrucción especifica con qué registro se opera. Ejemplo:

MOV Rn,A ; (Rn) ← (A)

- Direccionamiento implícito

Especifica implícitamente el registro sobre el que van a operar, como el acumulador, el puntero de datos, etc. No necesitan especificar el operando porque está implícito. Ejemplo:

INC A

- Direccionamiento inmediato

Después del código de operación se encuentra una constante en la memoria de programa. Ejemplo:

MOV A,#244 carga el acumulador con el número decimal 244.

- Direccionamiento indexado

Es utilizado para la lectura de tablas. Un registro de 16 *bits* (el DPTR o el contador de programa) apunta a la base que permite acceder a la lectura de esa posición de la tabla.

Ciclo de instrucción

Como hemos visto a lo largo del curso, una instrucción requiere de uno o más ciclos de máquina de acuerdo a su código de operación y al número de *bytes* que se maneje en cada instrucción. Por ejemplo, la instrucción **INC A** (**figura 10.1.a**) tiene un *byte* de instrucción y requiere un ciclo de máquina, la instrucción **INC DPTR** requiere dos ciclos de máquina (**figura 10.1.c**). De acuerdo al número de *bytes* que incorpore una instrucción para su funcionamiento, también varían los ciclos de máquina utilizados por la CPU para su ejecución. La instrucción **MOV A, #data** tiene dos *bytes* de instrucción y requiere de un ciclo de máquina (**figura 10.1b**), mientras que la instrucción **MOV direct, #data**, al constar de tres *bytes*, necesita de dos ciclos de máquina.

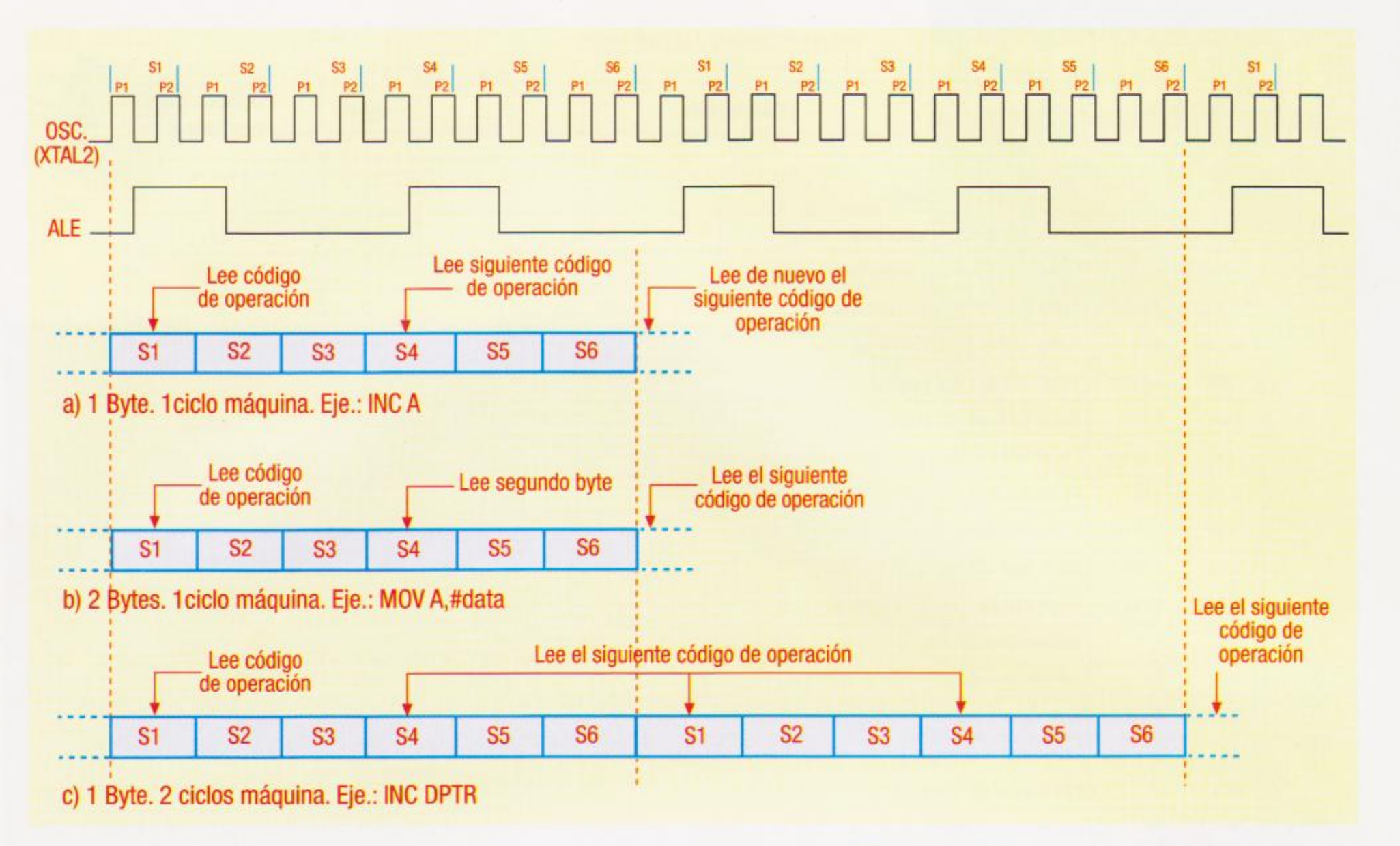


Figura 10.1 *Ciclos de máquina de las instrucciones*

Tipos de instrucciones

Como lo mencionamos, el conjunto de instrucciones del 8051 se puede dividir en categorías así:

- Instrucciones aritméticas
- Instrucciones lógicas
- Instrucciones de transferencia de datos
- Instrucciones booleanas
- Instrucciones de salto

Analicemos las principales características de cada categoría de instrucciones haciendo un seguimiento por medio de tablas. Para leer correctamente las tablas que se muestran a continuación es importante aclarar el significado de las convenciones utilizadas:

- Rn. Registros R0-R7 del banco de registros seleccionado.

- Directo. Dirección del dato de 8 *bits* de la memoria interna. Puede ser un dato de la RAM interna (0-127 para el 8051 ó 0-255 para el 8052), o un registro del FSR.

- @Ri. Se refiere a los registros R0 y R1, que permiten el direccionamiento indirecto para acceder a los datos de 8 *bits* de la RAM interna.

- #data. Operando constante de 8 *bits*.

- #data16. Operando constante de 16 *bits*.

- addr16. Dirección destino de 16 *bits*. Utilizada por las instrucciones de salto LCALL y LJMP para permitir el salto dentro del espacio de 64 *Kbytes* de la memoria de programa.

- addr11. Dirección destino de 11 *bits*. Utilizada por las instrucciones de salto ACALL y AJMP para permitir el salto dentro de la página de 2 *Kbytes* de la memoria de programa, a partir del primer *byte* de la siguiente instrucción.

- rel. Salto relativo en formato de 8 *bits* en complemento a 2. Utilizado por la instrucción SJMP y todos los saltos condicionales. El rango del salto (8 *bits*) está comprendido entre –128 a +127 *bytes*, a partir del primer *byte* de la siguiente instrucción.

INSTRUCCIONES ARITMÉTICAS

Nemónico		Descripción		Byte	Períodos de oscilador
ADD	A,Rn	Sumar registro al acumulador		1	12
ADD	A,direct	Sumar byte directo al acumulador	Dirección directa	2	12
ADD	A,@Ri	Sumar RAM indirecta al acumulador		1	12
ADD	A,#data	Sumar datos inmediatos al acumulador	Dato inmediato	2	12
ADDC	A,Rn	Sumar registro al acumulador con carry		1	12
ADDC	A,direct	Sumar byte directo al acumulador con carry	Dirección directa	2	12
ADDC	A,@Ri	Sumar RAM indirecta al acumulador con carry		1	12
ADDC	A,#data	Sumar dato inmediato al acumulador con carry	Dato inmediato	2	12
SUBB	A,Rn	Restar registros del acumulador		1	12
SUBB	A,direct	Restar byte directo del acumulador	Dirección directa	2	12
SUBB	A,@Ri	Restar RAM indirecta del acumulador		1	12
SUBB	A,#data	Restar dato inmediato del acumulador	Dato inmediato	2	12
INC	A	Incrementar acumulador		1	12
INC	Rn	Incrementar registro		1	12
INC	direct	Incrementar byte directo	Dirección directa	2	12
INC	@Ri	Incrementar RAM directa		1	12
DEC	A	Decrementar acumulador		1	12
DEC	Rn	Decrementar registro		1	12
DEC	direct	Decrementar byte directo	Dirección directa	2	12
DEC	@Ri	Decrementar RAM indirecta		1	12
INC	DPTR	Incrementar puntero de datos		1	24
MUL	AB	Multiplicar A y B		1	48
DIV	AB	Dividir A/B		1	48
DA	A	Ajuste decimal del acumulador		1	12

Tabla 10.1 *Instrucciones aritméticas*

Instrucciones aritméticas

Las podemos apreciar en la **tabla 10.1**. Principalmente podemos notar que se tienen instrucciones para multiplicar y para dividir. La instrucción **MUL AB** multiplica el contenido del acumulador con el dato situado en el registro B y el producto, en formato de 16 *bits*, aparece en los registros acumulador y B. La instrucción **DIV AB** divide el contenido del acumulador con el dato situado en B, apareciendo el resultado en el acumulador y el residuo en el registro B.

Instrucciones lógicas

En la **tabla 10.2** podemos apreciar el conjunto de las instrucciones lógicas. Las primeras instrucciones corresponden a las operaciones AND, OR, EXOR y NOT. Luego aparecen las instrucciones de manipulación de la información, como la rotación, por ejemplo.

Instrucciones de transferencia de datos

Podemos distinguir tres categorías dependiendo de que la transferencia se haga sobre la RAM interna, la RAM externa o la memoria de programa para manipular tablas.

En la **tabla 10.3** podemos observar las instrucciones pertenecientes a esta categoría.

Las primeras instrucciones permiten mover datos dentro de la RAM interna, de manera general podemos deducir fácilmente la operación de cada una de ellas haciendo una analogía con lo visto en las anteriores familias de microcontroladores, especialmente la Motorola. Dentro de la tabla se encuentran en el **bloque A**.

INSTRUCCIONES LÓGICAS

Nemónico		Descripción			Byte	Períodos de oscilador
ANL	A,Rn	AND entre el registro y el acumulador			1	12
ANL	A,direct	AND entre el byte directo y el acumulador	Dirección directa		2	12
ANL	A@Ri	AND entre RAM indirecta con el acumulador			1	12
ANL	A,#data	AND entre dato inmediato y el acumulador	Dato inmediato		2	12
ANL	direct,A	AND entre el acumulador y un byte directo	Dirección directa		2	12
ANL	direct,#data	AND entre dato inmediato y byte directo	Dirección directa		3	24
ORL	A,Rn	OR entre un registro y el acumulador			1	12
ORL	A,direct	OR entre byte directo y el acumulador	Dirección directa	Dato inmediato	2	12
ORL	A,@Ri	OR entre RAM indirecta y el acumulador			1	12
ORL	A,#data	OR entre dato inmediato y el acumulador	Dato inmediato		2	12
ORL	direct,A	OR entre el acumulador y un byte directo	Dirección directa		2	12
ORL	direct,#data	OR entre dato inmediato y byte directo	Dirección directa	Dato inmediato	3	24
XRL	A,Rn	OR exclusiva entre un registro y el acumulador			1	12
XRL	A,direct	OR exclusiva entre byte directo y el acumulador	Dirección directa		2	12
XRL	A,@Ri	OR exclusiva entre RAM indirecta y el acumulador			1	12
XRL	A,#data	OR exclusiva entre dato inmediato y el acumulador	Dato inmediato		2	12
XRL	direct,A	OR exclusiva entre acumulador y byte directo	Dirección directa		2	12
XRL	direct,#data	OR exclusiva entre dato inmediato y byte directo	Dirección directa	Dato inmediato	3	24
CRL	A	Borra acumulador			1	12
CPL	A	Complementar acumulador			1	12
RL	A	Rotar acumulador a la izquierda			1	12
RLC	A	Rotar acumulador a la izquierda a través del carry			1	12
RR	A	Rotar acumulador a la derecha			1	12
RRC	A	Rotar acumulador a la derecha a través del carry			1	12
SWAP	A	Intercambiar los nibbles en el acumulador			1	12

Tabla 10.2 *Instrucciones lógicas*

El segundo grupo de instrucciones son las relacionadas con transferencia de datos sobre la RAM externa. En la tabla se encuentran en el bloque que corresponde a la letra B. Las facilidades que ofrecen los microcontroladores Intel para acceder a la memoria RAM externa los convierte en dispositivos sumamente potentes para ser incorporados en tareas especializadas de instrumentación y control industrial.

Para el manejo de tablas existen dos instrucciones especializadas que facilitan esta labor. El nemónico es MOVC (mover constante).

Instrucciones booleanas

La familia de microcontroladores 8051 tiene un procesador booleano incluido en el circuito integrado. Con esta posibilidad es viable trabajar con el estado de los *bits* de algunos registros específicos de la memoria.

El conjunto de instrucciones que permiten este tipo de proceso, tan adecuado para tareas industriales y de comunicaciones, se encuentra relacionado en la **Tabla 10.4**.

Instrucciones de salto

Como hemos estudiado en lecciones anteriores, a veces se puede alterar la secuencia de ejecución normal de un programa para saltar a otras porciones del mismo. El salto puede ser condicional o incondicional y las instrucciones que lo permiten en el 8051 se encuentran resumidas en la **tabla 10.5**.

INSTRUCCIONES DE TRANSFERENCIA DE DATOS

	Nemónico		Descripción		Byte	Períodos de oscilador
A	MOV	A,Rn	Mover registro al acumulador		1	12
A	MOV	A,direct	Mover byte directo al acumulador	Dirección directa	2	12
A	MOV	A,@Ri	Mover RAM indirecta al acumulador		1	12
A	MOV	A,#data	Mover dato inmediato al acumulador	Dato inmediato	2	12
A	MOV	R,nA	Mover acumulador a registro		1	12
A	MOV	Rn,direct	Mover byte directo al registro	Dirección directa	2	24
A	MOV	Rn,#data	Mover dato inmediato al registro	Dato inmediato	2	12
A	MOV	direct,A	Mover acumulador al byte directo	Dirección directa	2	12
A	MOV	direct,Rn	Mover registro al byte directo	Dirección directa	2	24
A	MOV	direct,direct	Mover byte a posición directa	Dirección directa (fuente) / Dirección directa (destino)	3	24
A	MOV	direct,@Ri	Mover RAM indirecta al byte directo	Dirección directa	2	24
A	MOV	direct,#data	Mover dato inmediato a byte directo	Dirección directa / Dato inmediato	3	24
A	MOV	@Ri,A	Mover el acumulador a RAM indirecta		1	12
A	MOV	@Ri,direct	Mover byte a RAM indirecta	Dirección directa	2	24
A	MOV	@Ri,#data	Mover dato inmediato a RAM indirecta	Dato inmediato	2	12
A	MOV	DPTR,#data16	CArga puntero de dato con una constante de 16 bits	Dato inmediato 15-8 / Dato inmediato 7-0	3	24
B	MOVX	A,@Ri	Mover RAM externa al acumulador		1	24
B	MOVX	@Ri,A	Mover el acumulador a RAM externa		1	24
	PUSH	direct	Poner byte en la pila	Dirección directa	2	24
	POP	direct	Tomar byte de la pila	Dirección directa	2	24
	XCH	A,Rn	Intercambiar registro con acumulador		1	12

Tabla 10.3 *Instrucciones de transferencia de datos*

INSTRUCCIONES BOOLEANAS

Nemónico		Descripción		Byte	Períodos de oscilador
CLR	C	Borrar carry	1100 0011	1	12
CLR	bit	Borrar bit	1100 0010 / Dirección del bit	2	12
SETB	C	Activar carry	1101 0011	1	12
SETB	bit	Activar bit	1101 0010 / Dirección del bit	2	12
CPL	C	Complementar carry	1011 0011	1	12
CPL	bit	Complementar bit	1011 0010 / Dirección del bit	2	12
ANL	C,bit	AND entre un bit y el carry	1000 0010 / Dirección del bit	2	24
ANL	C,/bit	AND entre el carry y el complemento de un bit	1011 0000 / Dirección del bit	2	24
ORL	C,bit	OR entre un bit y el carry	0111 0010 / Dirección del bit	2	24
ORL	C,/bit	OR entre el carry y el complemento de un bit	1010 0000 / Dirección del bit	2	24
MOV	C,bit	OR entre un bit y el carry	1010 0010 / Dirección del bit	2	12
MOV	bit,C	OR entre el carry y el complemento de un bit	1001 0010 / Dirección del bit	2	24
JC	rel	Mover un bit al carry	0100 0000 / Dirección relativa	2	24
JNC	rel	Mover carry a un bit	0101 0000 / Dirección relativa	2	24
JB	bit,rel	Salte si el carry está activado	0010 0000 / Dirección del bit / Dirección relativa	3	24
JNB	bit,rel	Salte si el bit no es uno	0011 0000 / Dirección del bit / Dirección relativa	3	24
JBC	bit,rel	Salte si el bit es uno y bórrelo	0001 0000 / Dirección del bit / Dirección relativa	3	24

Tabla 10.4 *Instrucciones booleanas*

INSTRUCCIONES DE SALTO

Nemónico		Descripción			Byte	Períodos de oscilador
ACALL	addr11	Llamado absoluto a subrutina			2	24
LCALL	addr16	Llamado largo a subrutina			3	24
RET		Retornar de subrutina			1	24
RETI		Retornar de interrupción			1	24
AJMP	addr11	Salto absoluto			2	24
LJMP	addr16	Salto largo			3	24
SJMP	rel	Salto corto	Dirección relativa		2	24
JZ	rel	Salte si el acumulador es cero	Dirección relativa		2	24
JNZ	rel	Salte si el acumulador no es cero	Dirección relativa		2	24
CJNE	A,direct,rel	Compare un byte con el acumulador y salte si no son iguales	Dirección directa	Dirección relativa	3	24
CJNE	A,#data,rel	Compare el dato inmediato con un registro y salte si no son iguales	Dato inmediato	Dirección relativa	3	24
DJNZ	Rn,rel	Decremente registro y salte si no es cero	Dirección relativa		2	24
DJNZ	direct,rel	Decremente un byte y salte si no es cero	Dirección directa	Dirección relativa	3	24
NOP		No operación			1	12

Tabla 10.5 *Instrucciones de salto*

Los saltos incondicionales se ejecutan con la instrucción **JMP dirección**, de la cual se tienen tres variantes SJMP, LJMP y AJMP.

Los saltos condicionales se denominan así porque dependen del estado del registro de estado (PSW) o del estado de uno o más *bits* de la entrada/salida de periféricos.

Modelo de Programación

El término **Modelo de Programación** se entiende como el conjunto de registros accesibles por el usuario, que unidos a las instrucciones y los modos de direccionamiento, constituyen la materia prima para que un programador de microcontroladores pueda llevar a cabo su trabajo.

Vamos a estudiar ahora algunos ejercicios básicos de programación con los microcontroladores Intel. Podremos notar que teniendo claras las bases teórico-prácticas de una familia de microcontroladores, resulta fácil trabajar con otra arquitectura o tecnología.

Ejercicio No. 19
Estructura condicional

Una de las características más interesantes que se tiene en la programación de microcontroladores, es la posibilidad de tomar decisiones dependiendo del contenido de una posición de memoria. Es lo que se denomina una **bifurcación condicional**.

```
ORIGIN 0200H
SYMBOLS      ON
CODE

VBLE3        EQU        77H ; ASIGNACIÓN DE POSICIONES DE
VBLE2        EQU        78H ; MEMORIA PARA LAS DISTINTAS
VBLE1        EQU        79H ; VARIABLES QUE HACEN PARTE
RTA          EQU        7AH ; DEL EJERCICIO

             MOV   VBLE1, #01H ; ASIGNACIÓN DE VALORES
             MOV   VBLE2, #02H ; ARBITRARIOS A CADA VARIABLE
             MOV   A,     VBLE3 ; LEE VARIABLE VBLE3
             JNZ   SALIR
             MOV   A,     VBLE1  ; SI VBLE3 ES CERO SE EJECUTA
             ADD   A,     VBLE2  ; LA SUMA DE LOS DOS OPERANDOS
             MOV   RTA,   A      ; Y EL RESULTADO QUEDA EN VBLE3
             MOV   A,     VBLE3
             JZ    SALIR2

SALIR:
             MOV   A,     VBLE1  ; SI VBLE3 NO ES CERO, SE
             SUBB  A,     VBLE2  ; HACE LA RESTA Y GUARDA EL
                                 ; RESULTADO
             MOV   RTA,   A

SALIR2:
             NOP
             END
```

Figura 10.2 *Ejercicio sobre estructuras condicionales*

En este ejercicio, **figura 10.2**, observaremos este caso. Vamos a hacer una suma o una resta de dos variables VBLE1 y VBLE2. Si el contenido de un registro VBLE3 es cero, se suma el contenido de VBLE1 y VBLE2; de lo contrario se restan. El contenido de la operación que se ejecute se almacena finalmente en el registro RTA.

Ejercicio No. 20
Retardo por *software*

A lo largo de toda la obra hemos analizado el desarrollo de este tipo de retardos y los hemos incorporado dentro de nuestros proyectos. Es indudable que a pesar de que los retardos de mayor precisión son los generados por los temporizadores internos (el 8051 cuenta con dos temporizadores) y que ellos no impiden a la CPU mantener el control de los periféricos, los retardos por programa siguen siendo muy útiles en algunos casos.

Recordemos que para ejecutar este tipo de retardos procedemos a cargar algunos registros, entre más registros carguemos, mayor será el retardo. Luego entramos a implantar una serie de ciclos anidados donde secuencialmente se decrementan estos registros. En este ejercicio se cargan los registros R0, R1 y R2, como se muestra en el diagrama de la **figura 10.3** y en el código fuente de la **figura 10.4**.

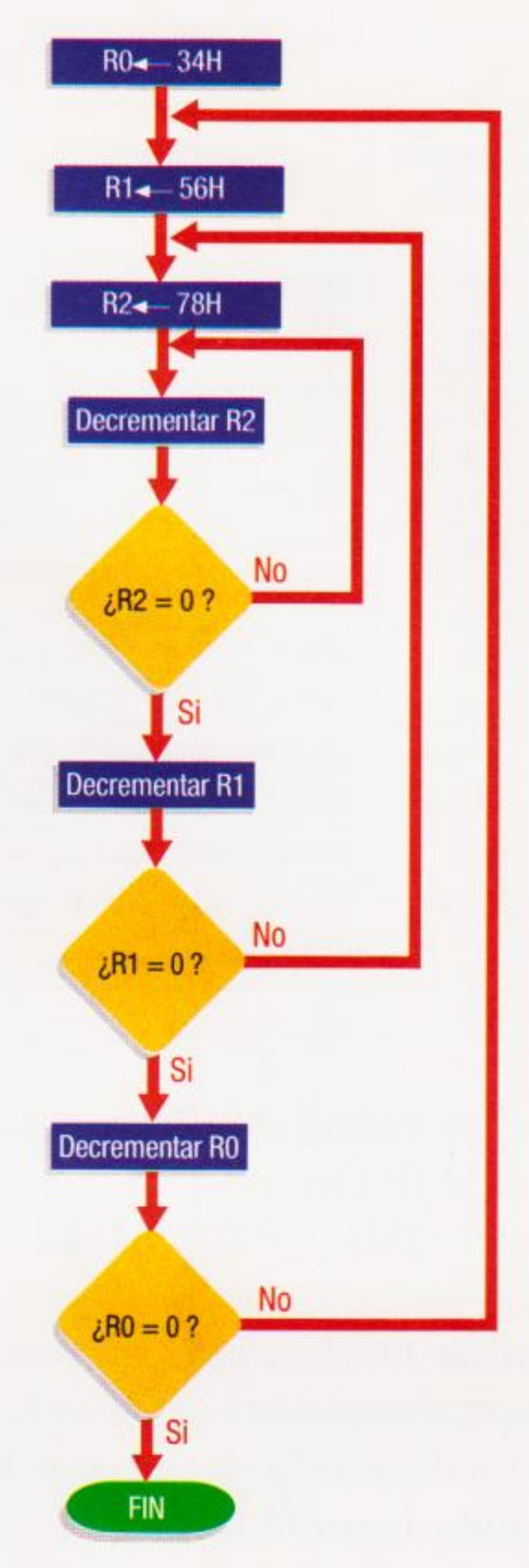


Figura 10.3 *Diagrama de flujo del ejercicio sobre retardo por programa*

```
        TITLE       RETARDO POR SOFTWARE
        COMMENT     * EJEMPLO BÁSICO DE DESARROLLO DE RETARDOS QUE DEPENDEN DE LOS
                      VALORES QUE SE CARGUEN EN LOS REGISTROS R0, R1 Y R2.
                    *

        ORIGIN      0200H
        SYMBOLS     ON
        CODE
        MOV   R0, #34H     ;SE CARGA EL REGISTRO R0 CON EL NÚMERO 34H
ET1:  MOV   R1, #56H       ;SE CARGA EL REGISTRO R1 CON EL NÚMERO 56H
ET2:  MOV   R2, #78H       ;SE CARGA EL REGISTRO R2 CON EL NÚMERO 78H
ET3:  DJNZ  R2, ET3        ;DECREMENTA R2 Y REPITE LA OPERACIÓN HASTA QUE R2 SEA CERO
        DJNZ  R1, ET2      ;DECREMENTA R1 Y REPITE LA OPERACIÓN HASTA QUE R1 SEA CERO.
                           ;POR CADA LAZO DE R1, SE EJECUTA 78 VECES EL LAZO R2
        DJNZ  R0, ET1      ;DECREMENTA R0 Y REPITE LA OPERACIÓN HASTA QUE R0 SEA CERO. POR
                           ;CADA LAZO DE R0, SE EJECUTA 56 VECES EL LAZO R1 Y 56 * 78
                           ;VECES R2.
        NOP
        END
```

Figura 10.4 *Ejercicio sobre retardo por programa*

Ejercicio No. 21
Subrutinas

Como hemos visto, un elemento fundamental que debe manejar todo buen programador de microcontroladores es la correcta agrupación de bloques de programa repetitivos en subrutinas. Ellas ahorran la cantidad de memoria utilizada y permiten una mayor claridad y comprensión de los códigos, lo que facilita su revisión o ampliación posterior. Como hemos estudiado en anteriores ocasiones, las subrutinas pueden ubicarse dentro del mismo programa principal o dentro de un archivo adicional que se pueda incluir desde el cuerpo principal.

En este ejercicio vamos a escribir en posiciones de memoria consecutivas, a partir de la 60H, el término "2002", incluyendo entre cada uno de sus caracteres un guión (-).

En la **figura 10.5** podemos observar el diagrama de flujo de este ejercicio y en la **figura 10.6** su código fuente. Podemos darnos cuenta que el programa principal y la subrutina conforman un único programa. El programa principal empieza en la dirección 0200H y la subrutina en la 0300H.

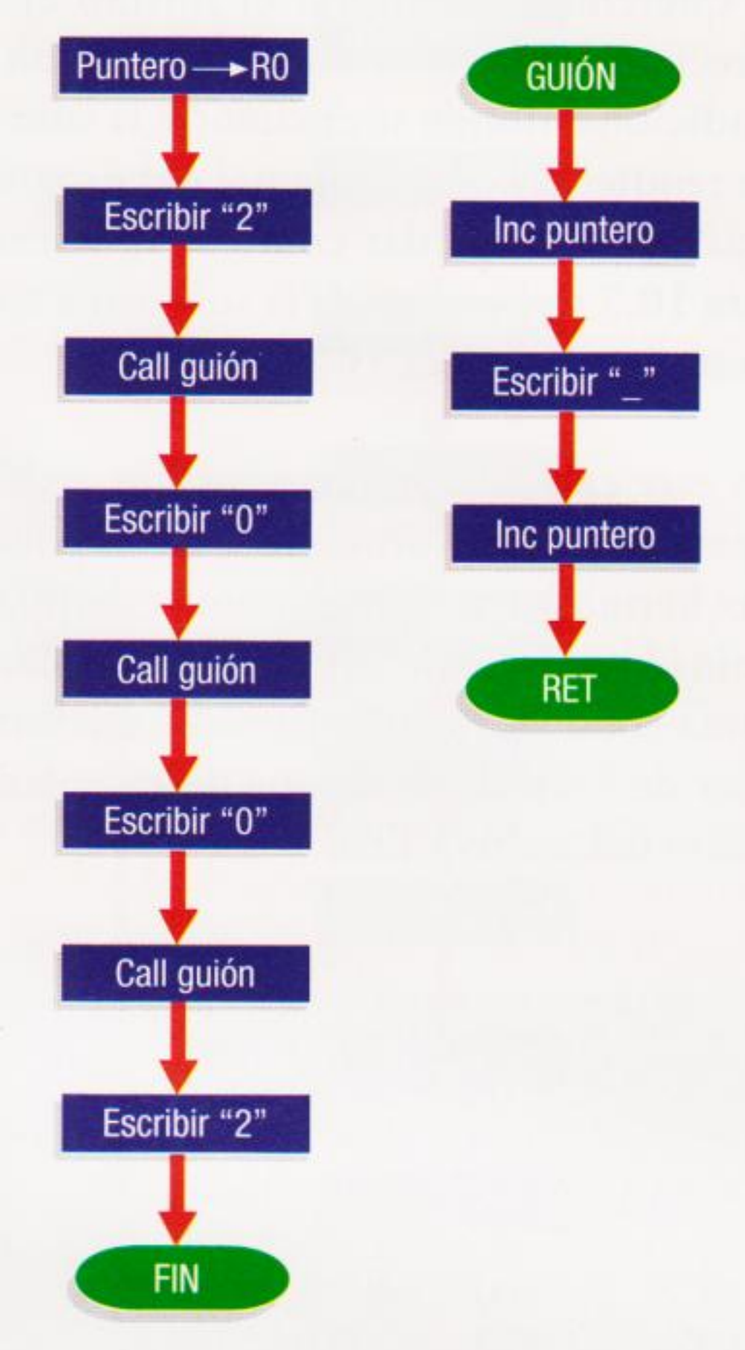


Figura 10.5 *Diagrama de flujo del ejercicio*

```
TITLE       Ejercicio de manejo de subrutinas
COMMENT     * Escribe caracteres y los separa con un guión.
                    El guión es una subrutina.
            *
ORIGIN      0200H
SYMBOLS     ON
CODE

PUNT        EQU   60H
GUION       EQU   0300H
            MOV   R0,   PUNT   ; INICIALIZA EL APUNTADOR
            MOV   @R0,  #32H   ; Carga en la posición de memoria PUNT el
                               ; carácter decimal "2"
            CALL  GUION
            MOV   @R0,  #30H   ; Carga en la posición de memoria PUNT + INC el
                               ; carácter decimal "0"
            CALL  GUION
            MOV   @R0,  #30H   ; Carga en la posición de memoria PUNT + INC el
                               ; carácter decimal "0"
            CALL  GUION
            MOV   @R0,  #32H   ; Carga en la posición de memoria PUNT + INC el
                               ; carácter decimal "2"
ORIGIN      0300H
            INC   R0
            MOV   @R0,  #2DH   ; Carga en la posición de memoria PUNT + INC el
                               ; carácter decimal "-"
            INC   R0           ; Prepara la posición de memoria para depositar el
                               ; siguiente
            RET                ; Retorno al programa principal
            END
```

Figura 10.6 *Código fuente del ejercicio*

Si queremos desarrollar el mismo ejercicio pero con la caraterística de incluir un archivo adicional donde se encuentre la subrutina que requiere el código fuente del programa principal, podría quedar como se muestra en la **figura 10.7** y el código de la subrutina como se muestra en la **figura 10.8**.

En este caso se compilan los dos archivos por separado y posteriormente se unen por medio una herramienta del programa compilador denominada enlazador (*linker*). Al ejecutar el programa enlazador el usuario debe escribir los nombres de los programas que desea enlazar y el nombre del archivo final.

Ejercicio No. 22
Manejo de la pila y de las rotaciones

Para terminar con esta serie de ejercicios sobre microcontroladores Intel, vamos a estudiar brevemente el manejo de instrucciones de acceso a la memoria de la pila.

En los microcontroladores Intel es posible acceder a la pila por voluntad del programador, para guardar valores importantes. No ocurre lo mismo en otras familias de microcontroladores, donde la pila se limita únicamente a almacenar automáticamente datos cuando se presenta un salto a una subrutina o una interrupción. El ejercicio consiste en que un dato desarrolle una

```
            ORIGIN     0200H
            SYMBOLS    ON

            CODE

PUNT        EQU   60H
MENOS       EQU   0300H
            MOV   R0, #PUNT  ;inicialización del puntero
            MOV   @R0, #32H  ;Carga en PUNT el carácter 2
            CALL  MENOS      ;Llama la subrutina MENOS
            MOV   @R0, #30H  ;Carga en PUNT el carácter 0
            CALL  MENOS      ;Llama la subrutina MENOS

            MOV   @R0, #30H  ;Carga en PUNT el carácter 0
            CALL  MENOS      ;Llama la subrutina MENOS

            MOV   @R0, #32H  ;Carga en PUNT el carácter 2
            CALL  MENOS      ;Llama la subrutina MENOS

            NOP
            END
```

Figura 10.7 *Archivo principal*

```
ORIGIN     0300H
SYMBOLS    ON
CODE

INC   R0          ;Incrementa R0 = PUNT + 1
MOV   @R0, #2DH   ;Carga en PUN + INC el carácter "-"
INC   R0
RET               ;Retorno al programa principal
END
```

Figura 10.8 *Código de la subrutina en un archivo separado*

Figura 10.9 *Ilustración del ejercicio de rotación de un registro*

rotación completa y que finalice en la misma posición con la que empezó. Para cumplir con este objetivo se deben llevar a cabo ocho rotaciones, **figura 10.9**.

Una de las instrucciones más utilizadas es la de rotar a través del acarreo (*carry*), pero en este caso no haremos uso de ella. Utilizaremos la pila debido a que el registro de rotación y el contador de rotaciones que se necesitan para este ejercicio usan el mismo registro físico, el acumulador (ACC); el diagrama de flujo lo podemos apreciar en la **figura 10.10** y el código en la **figura 10.11**.

Con esto hemos finalizado los ejercicios con microcontroladores Intel. Una de las mejores recomendaciones que se pueden seguir para mejorar en el mundo de la programación de microcontroladores, es escribir secciones de programa y subrutinas de manera que puedan ser utilizadas más adelante dentro de otros programas.

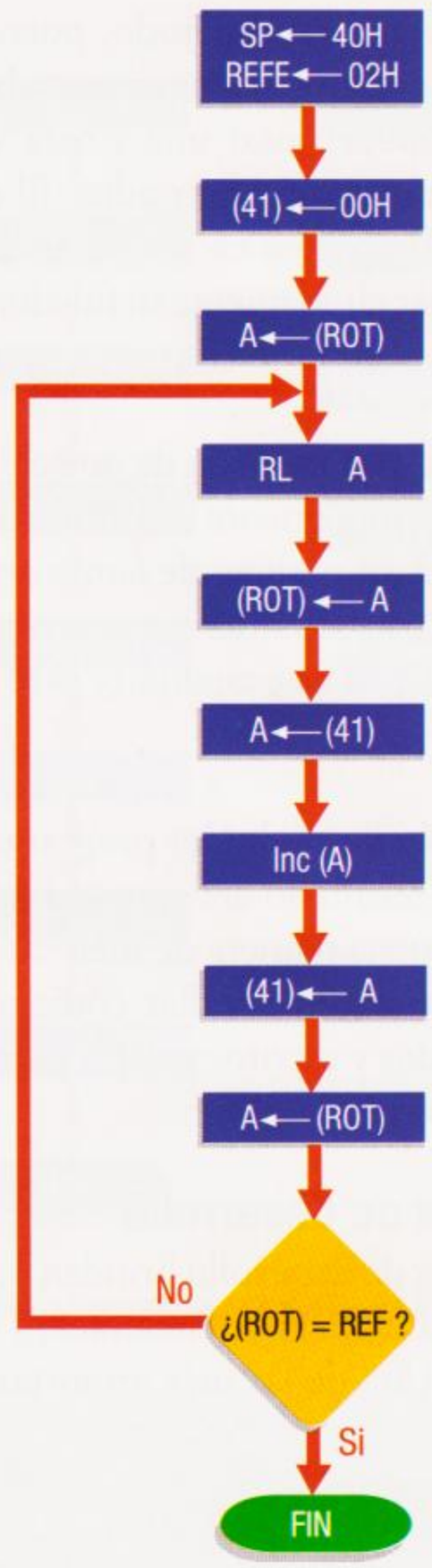


Figura 10.10 *Diagrama de flujo del ejercicio de manejo de la pila*

```
        ORIGIN      0200H
        SYMBOLS     ON
        CODE
REFE    EQU     70H
ROTA    EQU     6FH
        MOV     SP,#40H             ;Inicialización de la pila
        MOV     A, #00H             ;Se carga el acumulador con 00
        PUSH    A                   ;Se guarda en la pila
        MOV     REFE, #04H          ;Se inicializa el registro referencia
        MOV     ROTA, REFE          ;Se inicializa el registro de rotación
        MOV     A, ROTA             ;Se carga el acumulador con el valor de ROTA
SAL0:   RL      A                   ;El acumulador se rota a la izquierda
        MOV     ROTA, A             ;Una vez rotado se guarda
        POP     A                   ;Recupera el valor del contador. Ahora el
        INC     A                   ;acumulador es contador
        PUSH    A                   ;Guarda contador en la pila
        MOV     A, ROTA             ;Recupera el dato rotado
        CJNE    A, REFE, SAL0       ;Compara el acumulador con la referencia para
        NOP                         ;salir del programa cuando sean iguales.
        END
```

Figura 10.11 *Código fuente del ejercicio*

Teniendo en cuenta este método, poco a poco podremos estructurar una buena cantidad de librerías que se pueden usar una y otra vez para construir programas más avanzados. El elemento clave es establecer una excelente documentación para recordar rápidamente su funcionamiento y manejo.

Aunque existe la creencia de que el lenguaje ensamblador es simplemente la configuración secuencial de registros y trucos de *hardware*, la verdad es que él puede entenderse como un lenguaje estructurado igual que cualquier otro lenguaje de alto nivel.

La estructura de cualquier programa en ensamblador es el resultado del pensamiento de su creador y una buena manera de mejorar nuestro estilo de programar es estudiar códigos fuente bien estructurados y escritos por programadores con amplia experiencia.

Herramientas de desarrollo

Las herramientas de desarrollo brindan una enorme ventaja para agilizar la elaboración de productos finales. Una de las más importantes son los emuladores.

Estos dispositivos permiten ejecutar un programa paso a paso mientras se observa el contenido de los registros, por lo que se pueden detectar y corregir los errores de programación.

En este curso hemos estudiado muy superficialmente la familia de microcontroladores 8051, a pesar de ser tan poderosos hay que señalar que son algo antiguos. Su éxito ha radicado en que han sido catalogados con el apelativo de "estándar industrial". Si alguna vez el lector se decide por esta familia de microcontroladores, debe recordar que existen algunos derivados del 8051 que son muy interesantes, como los SAB80C535 y SAB80C537 de Siemens, el MSM80C154 de OKI y los 83C552, 87C750, 83CL782 y 83C851, de Phillips.

En Internet se puede encontrar una muy buena cantidad de herramientas de desarrollo para los microcontroladores Intel. Uno de ellos es el **NoICE Remote Debugger**, un programa tipo *shareware* elaborado por el ingeniero americano John Hartman. Cuenta con una tarjeta que incluye una ROM con un programa monitor que se comunica con la PC via RS-232 para depurar el programa del usuario cargado en la RAM de la computadora, **figura 10.12**.

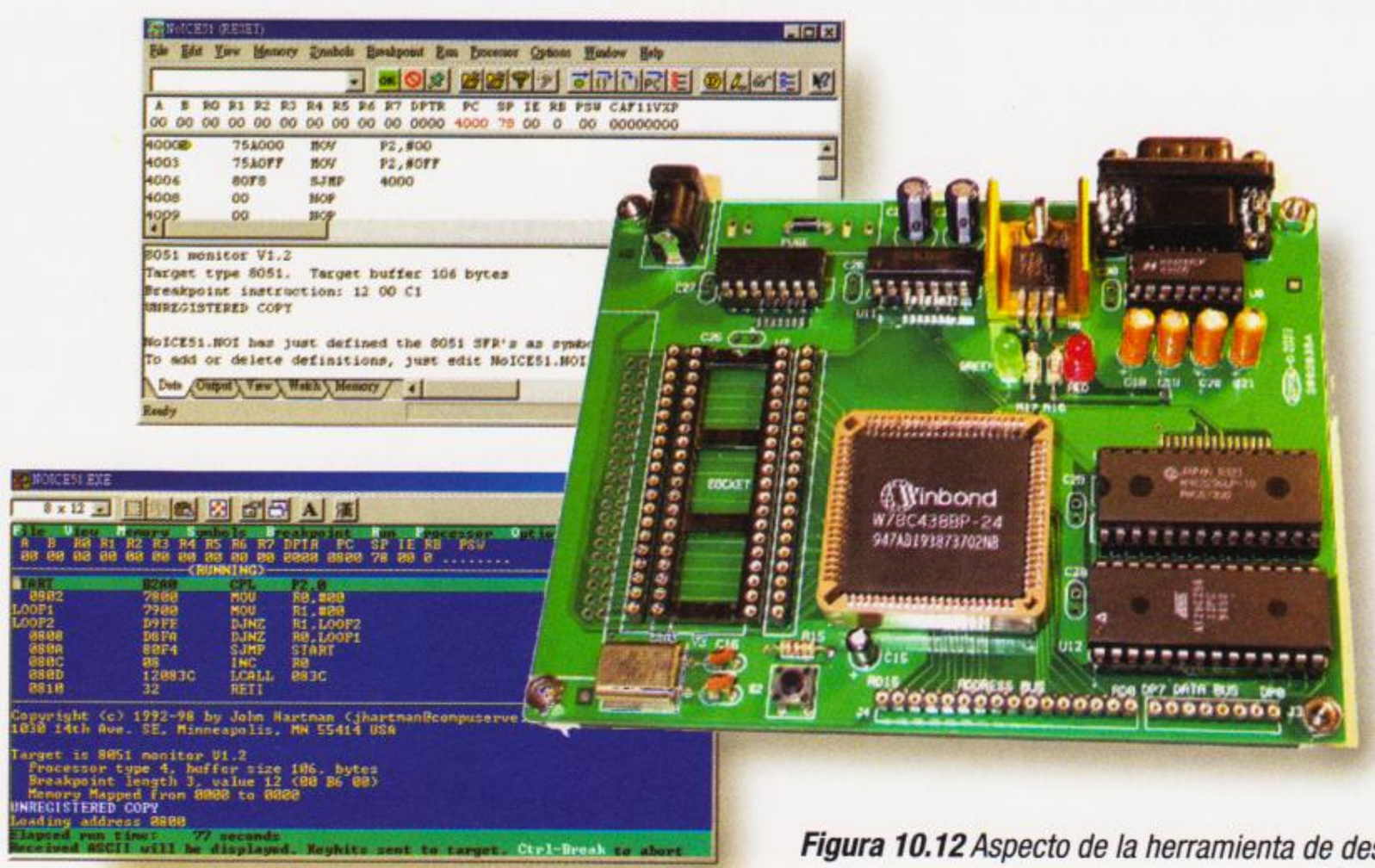
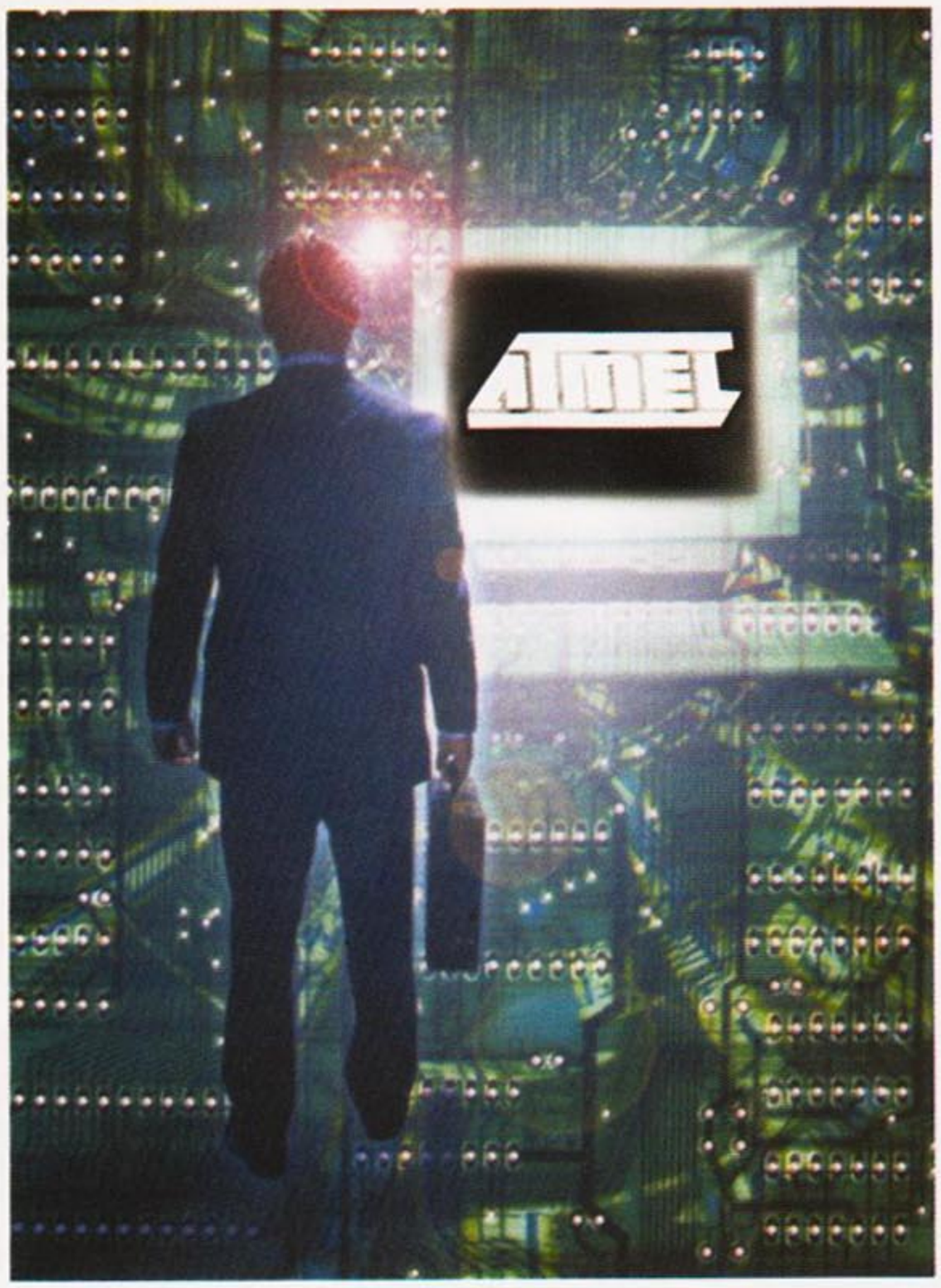


Figura 10.12 *Aspecto de la herramienta de desarrollo NoICE Remote Debugger*

LECCIÓN 11



PROGRAMACIÓN DE MICROCONTROLADORES ATMEL

En esta sección estudiaremos algunos aspectos fundamentales de la programación de estos dispositivos, los cuales, nos darán claridad para iniciarnos en el mundo de estos maravillosos microcontroladores.

Manejo de registros

Ya conocemos cómo se estructura un registro de ocho *bits* y cómo interpretar el valor numérico que representa una cantidad en este formato.

En los microcontroladores AVR existen treinta y dos (32) registros, nombrados R0, R1.... R31; el usuario puede nombrar cada registro a su gusto por medio de una directiva, así:

```
.DEF Mi_registro = R16
```

con la anterior línea podemos llamar, dentro de un programa, al registro Mi_registro en lugar de R16, así:

```
LDI Mi_registro, 100
```

lo que significa que se carga el número 100 en el registro R16.

Al igual que en los microcontroladores Intel, los AVR permiten mover directamente los contenidos de una posición de memoria a otra, así:

```
.DEF Mi_registro = R16
.DEF Otro_registro = R11
LDI Mi_registro, 100
MOV Otro_registro, Mi_registro
```

Hay que tener mucho cuidado, solo los registros R16 a R31 están en capacidad de cargar valores constantes con el comando LDI, del R0 al R15 no pueden hacer esta tarea. Lamentablemente esta restricción no se pudo evitar durante la construcción del conjunto de instrucciones para los AVR. Existe una excepción para esta regla, la instrucción CLR **Cualquier_registro** pone cualquier registro en cero.

La misma restricción se cumple para los siguientes comandos:

- ANDI Rx,K ; operación lógica AND entre el registro Rx y un valor constante K.
- CBR Rx,M ; borra todos los *bits* del registro Rx que se encuentran en uno enmascarando con M.
- CPI Rx,K; compara el contenido del registro Rx con un valor constante K.
- SBCI Rx,K; resta la constante K y el valor actual del acarreo del contenido del registro Rx. El resultado lo almacena en Rx.
- SBR Rx,M; coloca todos los *bits* del registro Rx en uno, de acuerdo a los *bits* que están en uno en el registro máscara M.
- SER Rx; coloca todos los *bits* del registro Rx en uno. Equivale a LDI Rx,255.
- SUBI Rx,K; resta el contenido de K del contenido del registro Rx. El resultado lo almacena en Rx.

En todos estos comandos el registro debe estar entre R16 y R31. Si planea usar uno de ellos debe seleccionar uno de los anteriores registros. Una de las ventajas de nombrar cada uno de los registros es que se facilita cambiar más adelante la posición de cualquiera de ellos.

Registros apuntadores

Una característica muy particular cumplen los pares de registros R26:R27, R28:R29 y R30:R31. Este conjunto de pares poseen nombres extras en ensamblador, ellos son: X, Y, y Z. Estos pares son registros apuntadores de 16 *bits*, y son capaces de direccionar posiciones de 16 *bits* en SRAM (X, Y o Z) o posiciones de la memoria de programa (Z).

El *byte* bajo de la dirección de 16 *bits* se encuentra en el registro más bajo y el *byte* alto en el registro más alto. Ambas partes tienen su nombre propio, por ejemplo, el *byte* alto de Z se denomina ZH (=R31) y el *byte* bajo es ZL (=R30). Estos nombres vienen definidos dentro de los archivos cabecera estándar que se incluyen para cada microcontrolador en particular. La división de estos apuntadores de 16 *bits* en dos *bytes* diferentes se hace así:

.EQU Adress = RAMEND ; RAMEND es la dirección de 16 *bits* más alta en SRAM

LDI YH,HIGH(Adress) ; configura el *byte* más significativo (MSB)

LDI YL,LOW(Adress) ; Configura el *byte* menos significativo (LSB)

El acceso por medio de apuntadores se programa mediante comandos especiales. Los accesos para lectura se denominan LD (**LoaD**) y los accesos para escritura se denominan ST (**STore**).

En los microcontroladores AVR solo se tiene un comando para leer el almacenamiento del programa. Éste se define para el apuntador Z y se denomina LPM (*Load from Program Memory*: Cargar desde la memoria de programa). El comando copia el *byte* ubicado en la dirección Z de la memoria de programa en el registro R0.

Como la memoria de programa posee dimensión de una palabra (un comando o una dirección constan de 16 *bits* ó 2 *bytes*) el *bit* menos significativo selecciona el *byte* bajo o alto (0=*byte* bajo, 1=*byte* bajo).

Debido a esto la dirección original debe multiplicarse por dos y los accesos se limitan a 15 *bits* ó 32 kB de la memoria de programa. Así:

```
LDI     ZH,HIGH(2*Dirección)
LDI     ZL,LOW(2*Dirección)
LPM
```

Después de este comando la dirección debe incrementarse para apuntar al siguiente *byte* en la memoria de programa. Como esta operación es muy utilizada, se ha implantado un comando especial para el incremento del apuntador:

```
ADIW ZL,1
LPM
```

ADIW significa que sume la palabra inmediata y un valor numérico que puede llegar hasta 63. Note que el programa ensamblador espera que se le entregue la parte baja del registro apuntador ZL como primer parámetro. Esto puede ser algo confuso debido a que es una operación en 16 *bits*.

Hay un comando complementario al anterior el cual resta un valor constante entre 0 y 63 del registro apuntador de 16 *bits* y se denomina **SBIW** (*Substract Inmediate Word*). ADIW y SBIW se pueden utilizar con los pares de registros X, Y y Z y el par de registros R25:R24, el cual no tiene un nombre adicional y no accede a posiciones de SRAM o memoria de programa. R25:R24 son ideales para manejar cantidades de 16 *bits*.

Una aplicación muy particular para los registros apuntadores es el acceso de registros a ellos mismos. Los registros se localizan en los 32 primeros *bytes* del espacio de direcciones del circuito integrado (0x0000 a 0x001F).

Este acceso se justifica cuando se necesita copiar el contenido del registro en SRAM ó en EEPROM.

Uno de los usos más comunes que se hace de los apuntadores es acceder a tablas de valores fijos en la memoria de programa. Por ejemplo, tenemos una tabla con 10 valores diferentes de 16 *bits* donde debemos leer el quinto elemento y almacenarlo en R25:R24:

```
Mi_Tabla:
.DW 0x1234,0x2345,0x3456,0x4568,0x5678
                          ; Los valores de la tabla
.DW 0x6789,0x789A,0x89AB,0x9ABC,0xABCD
                          ;

Leer5:
LDI ZH,HIGH(Mi_tabla*2)  ;Dirección de la tabla
                          ;al apuntador Z
LDI ZL,LOW(MyTable*2)    ;se multiplica por dos para
                          ;acceder al tamaño del byte
ADIW ZL,10                ;Apunta al quinto valor en
                          ;la tabla
LPM                       ;Lee el byte menos
                          ;significativo de la
                          ;memoria de programa
MOV R24,R0                ;Copia LSB al registro de
                          ;16 bits
ADIW ZL,1                 ;Apunta al MSB en la
                          ;memoria de programa
LPM                       ;Lee MSB
MOV R25,R0                ;Copia MSB al registro de
                          ;16 bits
```

Recomendaciones para el uso de registros

1. Hay que definir los nombres de los registros mediante la directiva .DEF, nunca usarlos con su nombre directo Rx.
2. Si necesita acceso a los apuntadores, utilice los registros R26 a R31 para ese propósito.
3. Los contadores de 16 *bits* póngalos en R25:R24.
4. Si necesita leer datos de la memoria de programa (por ejemplo para leer tablas), utilice Z (R31:R30) y R0.
5. Si planea acceder a *bits* de algunos registros (por ejemplo para leer banderas) utilice R16 a R23.

Los puertos en los microcontroladores AVR

En los microcontroladores AVR existen 64 puertos diferentes, los cuales no están físicamente disponibles para todos los tipos. Ellos lo estarán dependiendo del espacio de memoria y otros elementos de *hardware* interno. Los puertos poseen direcciones fijas con las cuales se comunica la CPU. Esa dirección es independiente del tipo de AVR; por ejemplo, el puerto B siempre tendrá la dirección 0x18.

Para trabajar con los puertos dentro de un programa, ellos deben ser definidos al inicio del mismo para lo cual se puede incluir un archivo con todas las definiciones, lo que, por lo general, proporciona el fabricante, así:

```
.EQU PORTB, 0x18
```

Para incluir por ejemplo, el archivo 8515.inc se escribe:

```
.INCLUDE«C:\8515def.inc»
```

La lectura del contenido de un puerto es posible mediante el comando IN. La secuencia:

```
.DEF MiRegistro = R16
IN MiRegistro, MCUCR
```

lee los *bits* del registro MCUCR y los lleva a MiRegistro.

Es muy normal que se deban tomar decisiones de acuerdo a la lectura de un puerto. En estos casos no se necesita leer todo un puerto y enmascarar *bits* para decidir sobre un *bit* en particular. Algunos comandos dan la oportunidad de ejecutar acciones dependiendo del nivel de algunos *bits*, esto lo veremos un poco más adelante. De la misma manera se pueden activar o desactivar *bits* específicos de un registro sin afectar a los demás. Los comandos SBI (Set Bit I/O) y CBI (Clear Bit I/O) cumplen con esta labor. Así:

```
.EQU BitActivo=0      ;El bit que se va  a
                      ;cambiar
SBI PortB, BitActivo  ;El bit que se va a
                      ;activar
CBI PortB, BitActivo  ;El bit que se va a
                      ;desactivar
```

Estas dos instrucciones tienen una limitante: solo los puertos con una dirección inferior a 0x20 pueden ser manejados. Los puertos también pueden accederse mediante comandos de acceso a SRAM, como LD y ST, simplemente sumando 0x20 a la dirección del puerto (las primeras 32 direcciones son los registros) así:

```
.DEF MiRegistro = R16
LDI ZH,HIGH(PORTB+32)
LDI ZL,LOW(PORTB+32)
LD MiRegistro,Z
```

La memoria SRAM

Todos los microcontroladores AT90S-AVR poseen RAM estática (SRAM). Las memorias SRAM se caracterizan porque no se pueden acceder directamente por la unidad de procesamiento (ALU, acumulador).

Para acceder a posiciones de esta memoria se utilizan registros intermedios. En el siguiente ejemplo un valor ubicado en SRAM se copiará en el registro R2 (primer comando), luego se hace una operación sobre R3, y el resultado regresa a R3, **figura 11.1**.

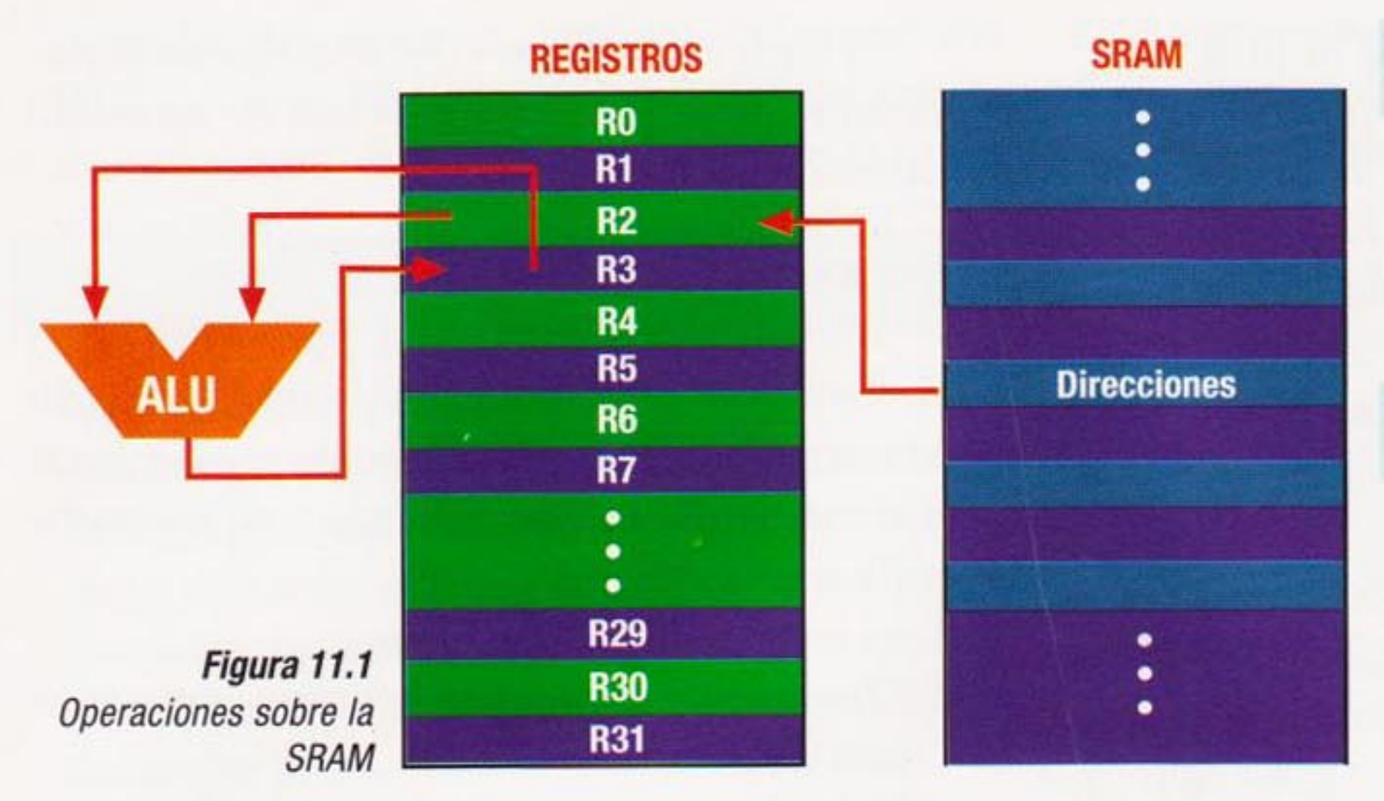


Figura 11.1
Operaciones sobre la SRAM

Es fácil deducir que las operaciones ejecutadas sobre valores en SRAM son más lentas que usando registros normales. La ventaja es que hasta el AVR más pequeño cuenta con 128 *bytes* de SRAM y sólo 32 registros normales.

Los AVR, desde el AT90S8515, ofrecen la posibilidad de conectar RAM externa, expandiendo los 512 *bytes* internos. Desde el punto de vista del lenguaje ensamblador la SRAM externa se accede de la misma manera que la SRAM interna. No se requieren otros comandos.

¿Para qué podemos usar la SRAM?

Además del simple almacenamiento de valores, la SRAM ofrece algunas buenas posibilidades. Se pueden utilizar apuntadores para llevar a cabo accesos flotantes a posiciones consecutivas. De esta manera podemos construir *buffers* circulares para el almacenamiento interno de valores. Esto no es posible con registros debido a que ellos son pocos y requieren de accesos fijos.

¿Cómo utilizar la SRAM?

Para copiar un valor en una posición de memoria SRAM, primero se debe definir la dirección. Las direcciones en la SRAM que se pueden utilizar se encuentran desde 0x0060 hasta el final de la SRAM, de cada circuito integrado en particular (por ejemplo en el AT90S8515 la máxima dirección es 0x025F). Mediante el comando:

```
STS     0x0060,     R1
```

el contenido del registro R1 se copia en la primera posición de la SRAM. Con la instrucción:

```
LDS     R1,     0x0060
```

el contenido de la posición 0x0060 de memoria SRAM se copia en un registro. Ésta es la manera como se lleva a cabo el acceso directo a una dirección definida por el programador.

Otra manera de acceder a la SRAM es mediante el uso de apuntadores. Se necesitan dos registros donde se almacena la dirección de 16 *bits* (X, Y y Z). Ellos permiten acceder directamente a la posición a que apunten, y, mediante incrementos o decrementos posteriores de los apuntadores, ejecutar un barrido de varias posiciones. Por ejemplo, un sencillo acceso a tres posiciones es así:

```
.EQU PosicionParaGuardar = 0x0060
.DEF MiRegistro =R1
.DEF OtroRegistro = R2
.DEF OtroRegistroMas = R3

LDI XH, HIGH(PosicionParaGuardar)
LDI XL, LOW(PosicionParaGuardar)
LD MiRegistro, X+
LD OtroRegistro, X+
LD OtroRegistroMas, X
```

Utilización de la SRAM como pila

El uso más común de la SRAM es como pila. Podemos pensar en la pila como en un edificio formado por pequeños bloques. Cada vez que llega un nuevo bloque se ubica en la cima del edificio; cuando se necesita tomar un valor debido a un retorno dentro del programa, se remueve el bloque que esté más alto en la cima del edificio. Esta estructura se denomina *Last-In-First-Out (LIFO)*, es decir: el último en llegar es el primero en salir.

Definiendo la SRAM como la pila

Para poder utilizar la SRAM como pila primero se debe configurar el puntero de pila, el cual es un apuntador de 16 *bits* que se puede acceder como si fuera un puerto normal. Este registro doble se denomina SPH:SPL; SPH almacena el *byte* más significativo de la dirección y SPL el menos significativo.

Esto solo se cumple si el AVR con el cual se trabaje posea más de 256 *bytes* de SRAM. De lo contrario el SPH no se encuentra definido y por lo tanto no debe usarse. En los siguiente ejemplos asumimos que contamos con más de 256 *bytes* de RAM.

Para construir la pila, el puntero a ella debe cargarse con la dirección de SRAM más alta que se encuentre disponible, así:

```
.DEF MiRegistro = R16
LDI MiRegistro, HIGH(FINRAM) ;byte más alto
OUT SPH,MiRegistro           ;al puntero a la pila
LDI MiRegistro, LOW(FINRAM) ;byte más bajo
OUT SPL,MiRegistro           ;al puntero a la pila
```

El valor de FINRAM depende del tipo de procesador, el cual se define en el archivo de definiciones incluido. Por ejemplo, el archivo *8515def.inc* tiene la línea:

```
.equ FINRAM =$25F ;última posición de SRAM
```

El archivo *8515def.inc* se incluye dentro del programa mediante la directiva

```
.INCLUDE "C:\8515def.inc"
```

al principio del código fuente. Así definimos la pila; ahora podemos despreocuparnos del puntero a la pila, pues sus manipulaciones se hacen automáticamente.

Utilización de la pila

Utilizar la pila es muy sencillo. Los contenidos de los registros se ponen en la pila así:

```
PUSH MiRegistro
```

El lugar a donde ese valor se dirige no es importante, pero, si necesitamos leer el contenido ejecutamos la instrucción:

```
POP MiRegistro    ;Lee el valor
```

Con POP tomamos el último valor ubicado en la cima de la pila. El proceso de poner y sacar los datos de ella es algo delicado y sólo se debe acudir a él cuando:

- El contenido se necesita de nuevo, algunas líneas de código después.
- Todos los registros se están utilizando.
- No se tienen más oportunidades de almacenar ese valor en otro lugar.

Si estas condiciones no están dadas, el uso de la pila para almacenar registros no es viable, y solo malgasta el tiempo del procesador.

La pila se utiliza en el llamado a subrutinas, donde se debe retornar a la dirección desde donde se llamó la subrutina. En este caso, cuando se invoca un programa, se toma la dirección de retorno (el valor actual del contador de programa) en la pila y salta a la subrutina.

Después de la ejecución se toma la dirección de retorno de la pila y se carga en el contador de programa. La ejecución del programa continúa exactamente una instrucción después del llamado.

```
RCALL ET1     ;Salta a la etiqueta ET1
[...] aquí se continúa con el programa
```

Éste es el salto a la etiqueta ET1 localizada en alguna parte del programa

```
ET1: ;ésta es la dirección de salto
[...] Aquí se hace algo
[...] finalizamos y ahora queremos
retornar a la posición llamada
RET
```

Durante la ejecución de una instrucción RCALL, automáticamente se incrementa el

contador de programa. Mediante la instrucción RET se carga el contenido anterior del contador de programa y la ejecución continúa desde ese punto.

Introducción a la programación de microcontroladores AVR

Analicemos ahora algunos aspectos fundamentales de la programación con microcontroladores AVR, como el manejo del *reset*, la temporización, las subrutinas y algunos comandos. Todos estos elementos nos permitirán tener una idea concreta de cómo desarrollar programas con esta tecnología.

Estado de *reset*

Cuando la fuente de alimentación alcanza un valor desde el cual el AVR empieza a trabajar, el circuito inicia una secuencia de *reset*, donde el contador de programa toma el valor inicial (cero) que corresponde a la dirección donde comienza el programa. Se puede llegar al estado de *reset* por varias circunstancias:

1. La presencia de una señal externa en el pin de *reset* del microcontrolador.
2. El *watchdog* ha alcanzado su máximo valor. El temporizador *watchdog* se debe reiniciar continuamente dentro del programa; de otra manera el procesador se reiniciará.
3. Por un salto directo a esa dirección del programa.

Para la segunda opción es necesario que el *watchdog* ya se encuentre habilitado por el programa. La habilitación y el borrado se hace mediante comandos. Por ejemplo, para regresar el contador a cero se ejecuta la instrucción **WDR**, para evitar así un estado de *reset*.

Después de la ejecución de un estado de *reset* el procesador ejecuta el código ubicado en la dirección 0000. Para decirle al compilador que nuestro código fuente empieza en este punto se utiliza una directiva especial, que se escribe al principio del programa, así:

```
.CSEG
.ORG 0000
```

La primera instrucción hace que el compilador se dirija a la sección del código. Lo que se encuentra a continuación se escribe en la memoria de programa del dispositivo. Las otras secciones de memoria donde se pueden escribir datos son la EEPROM y la SRAM:

```
.ESEG

.DSEG
```

La directiva ORG permite manipular las direcciones desde el código. Por ejemplo, para ubicar una tabla en una posición exacta se puede utilizar **ORG**.

Como el código siempre empieza en la dirección cero, esta posición también recibe el nombre de vector de *reset*. Después del vector de *reset*, las siguientes posiciones en el espacio de programa (direcciones 0x0001, 0x0002, etc.) son los vectores de interrupción.

Ellas son las posiciones a donde salta la ejecución cuando se presenta una interrupción interna o externa. Estas posiciones, denominadas vectores, son específicas para cada tipo de procesador y dependen del *hardware* disponible en cada caso.

Al usar interrupciones al principio del código, en el vector de *reset*, se debe ubicar un comando de salto para llegar a los otros vectores. Cada vector de interrupción debe mantener un comando de salto al respectivo servicio de interrupción. La secuencia típica para el inicio de un programa es así:

```
.CSEG
.ORG 0000
RJMP Inicio
RJMP ServIntRut1
[...] aquí ubicamos los otros comandos
del vector de interrupción
[...] y éste es un buen sitio para
ubicar las rutinas de interrupción
Inicio: ;Éste es el inicio del programa
[...] Aquí ubicamos el programa
principal
```

El comando **RJMP** origina un salto a la etiqueta **Inicio:** localizada algunas líneas abajo. Las etiquetas siempre empiezan en la primera columna del código fuente y terminan con un signo :. Las etiquetas que no cumplan con estas condiciones no serán tomadas por el compilador. Si falta alguna etiqueta se producirán errores de compilación.

Ejecución lineal del programa y saltos

La ejecución de un programa siempre se lleva a cabo en forma lineal, a menos que ocurra una interrupción o una instrucción de salto. Los saltos son muy utilizados, por ejemplo, si queremos construir un contador de 32 *bits* mediante los registros R1 a R4. El *byte* menos significativo en R1 se incrementa en una unidad, si R1 se desborda (255+1 = 0), debemos incrementar R2. Si R2 se desborda debemos incrementar R3, etc.

El incremento en una unidad se hace mediante la instrucción **INC**, y, si se presenta un sobreflujo al ejecutar **INC R1**, el *bit* cero del registro de estado se pondrá en uno. Si el *bit* de cero se pone en uno debemos ejecutar un incremento adicional de los otros registros. De la misma manera como se utilizó la bandera de cero, se pueden utilizar otros *bits* de estado para reaccionar a las distintas condiciones:

```
BRCC/BRCS:  Bandera de acarreo
BRSH:       Igual o mayor
BRLO:       Menor
BRMI:       Menos
BRPL:       Adición
BRGE:       Mayor o igual (con bit de signo)
BRLT:       Menor (con bit de signo)
BRHC/BRHS:  Bandera de sobreflujo
            de medio byte
BRTC/BRTS:  Bit T
BRVC/BRVS:  Bandera de complemento
            a dos
BRIE/BRID:  Interrupciones habilitadas
```

Temporización durante la ejecución de un programa

Para ejecutar una instrucción, los AVR emplean un ciclo del reloj conectado. Si el procesador posee un reloj de 4MHz una instrucción requiere 250 ns, y a 10 MHz solo requiere de 100ns.

Para llevar a cabo retardos por programa se puede acudir a ciclos anidados y a instrucciones de "no operación" **NOP**.

Macros

Es muy común encontrarse en la situación de tener que escribir secuencias de código idénticas en distintas partes de un programa.

Para ahorrarnos este trabajo podemos recurrir a los macros. Los macros son secuencias de código que se insertan dentro del código mediante su nombre.

Por ejemplo asumamos que necesitamos continuamente un retardo, podemos construir el macro así:

```
.MACRO Retardo1
NOP
NOP
NOP
NOP
.ENDMACRO
```

La definición del macro no produce código alguno. Éste se produce si se invoca al macro por su nombre:

```
[...] en alguna parte del código fuente
Retardo1
[...] el código continúa aquí
```

el resultado final son cuatro instrucciones **NOP** insertadas en el código en esta posición.

Subrutinas

Las subrutinas permiten ahorrar espacio en la memoria de programa si las comparamos con los macros; estas estructuras se componen de una secuencia de instrucciones que sólo se escriben una vez dentro del programa y que se encapsulan bajo un nombre para poder invocarse desde cualquier parte del programa.

El ejemplo típico de una subrutina es así:

```
Delay10:
NOP
NOP
NOP
RET
```

Las subrutinas siempre empiezan con una etiqueta para poder saltar a ellas. En el caso anterior se tienen tres instrucciones **NOP** y luego la instrucción **RET**. Esta subrutina emplea siete ciclos: tres para los **NOP** y cuatro para la instrucción **RET**.

Para invocar la subrutina se emplean tres ciclos adicionales:

```
[...] alguna parte del código fuente
RCALL Delay10
[...] luego en el código fuente
```

RCALL es un llamado relativo. El llamado se codifica como un salto relativo, la distancia existente entre el llamado y la subrutina la calcula el compilador. La instrucción RET retorna el control del programa hacia el punto desde donde se invocó la subrutina.

Si deseamos saltar directamente hacia algún punto del código debemos utilizar la instrucción RJMP, así:

```
[...] en algún punto del código fuente
RJMP Delay10
Return:
[...] luego en el código fuente
```

La rutina a la cual se saltó no usa la instrucción **RET**. Para retornar al punto desde donde se hizo el salto, se debe referenciar ese punto con una nueva etiqueta y utilizar otra instrucción de salto para regresar.

RCALL y RJMP son saltos incondicionales. Para saltar a alguna posición dependiendo de alguna condición, se deben combinar las condiciones con las instrucciones de salto. El llamado condicional de una subrutina se puede hacer con los comandos que se muestran a continuación. Si deseamos llamar una subrutina, dependiendo del valor de ciertos *bits* de un registro, debemos usar la siguiente secuencia:

```
SBRC R1,7        ;Salta la siguiente
                 ;instrucción si el bit 7 es 0
RCALL UpLabel    ;llamar la subrutina
```

La instrucción **SBRC** lee el *bit* x de un registro y se salta la instrucción que le sigue, si dicho *bit* es cero. La instrucción **RCALL** se ejecuta sólo si el *bit* 7 de R1 es uno, si desea la condición recíproca, es decir, que se ejecute **RCALL** si el *bit* siete es cero, se debe utilizar la instrucción **SBRS**.

Si se necesita que se salte una instrucción, cuando dos registros son iguales, se utiliza:

```
CPSE R1,R2          ;Compara R1 y R2, salta
                    ;si son iguales
RCALL Subrutina     ;llama una subrutina
```

Si deseamos saltar una instrucción, dependiendo del valor de un *bit* en un puerto, se utilizan las instrucciones **SBIC** o **SBIS**, las cuales leen el *bit* indicado en la misma línea, así:

```
SBIC PINB,0     ;Salta si el bit 0 del
                ;puerto B es 0
RJMP etiqueta   ;Salta a etiqueta
```

La instrucción **RJMP** se ejecuta solo si el *bit* 0 del puerto B se encuentra en alto.

Interrupciones

Muchas veces debemos reaccionar a condiciones del *hardware* u otros eventos, como por ejemplo, el cambio de nivel en un pin. Podemos desarrollar una especie de lazo en donde se pregunte constantemente el estado de dicho pin; es lo que se conoce como un *polling*. Si dentro del programa no se tienen más cosas que hacer, ésta representa una buena opción. Pero, si por ejemplo, la duración de los pulsos que va a leer es del orden de los microsegundos, este método puede no ser el más apropiado, es cuando se hace necesaria la incorporación de interrupciones.

Una interrupción se activa por alguna condición del *hardware*, la cual inicialmente debe estar habilitada. Para ello el primer paso es habilitar un *bit* del registro de estado, el cual corresponde al habilitador general de interrupciones, mediante la instrucción:

```
SEI ;Activar interrupciones
```

Cuando se presenta una condición que genera interrupción, el procesador ubica el valor actual del contador de programa en la pila para poder retornar al punto donde abandonó el programa y ejecuta las instrucciones ubicadas a partir del vector de interrupciones.

En la **figura 11.2** podemos apreciar un pequeño programa cuyo objetivo es hacer la rotación de la iluminación de un LED entre un conjunto de ellos conectado al puerto B. En este pequeño programa podemos apreciar varios de los apectos mencionados sobre los microcontroladores AVR.

Así damos por terminada la sección de programación de este curso. A lo largo de las distintas lecciones desarrollamos una serie de ejercicios y prácticas orientadas a dar claridad sobre los conceptos que se analizan en la sección de teoría. Esperamos haber solucionado una buena cantidad de dudas y haber sentado unas buenas bases para el desarrollo de productos electrónicos de cierto nivel de complejidad, que solucionen una necesidad que se presente en nuestro medio y, de esta manera, la electrónica se convierta en una alternativa viable de desarrollo económico.

```
;*****************************************************************************************
; Este programa hace que secuencialmente se encienda un solo LED de un conjunto conectado
; al puerto B.Conecte diodos LED desde el puerto B (por medio de resistencias limitadoras) a 5 voltios
;*****************************************************************************************
;***** dispositivo AT90S1200   ***** Definición de registros I/O
.equ   PORTB  =$18          ; registro del puerto b
.equ   DDRB   =$17          ; registro de direcciÛn del puerto b
.equ   PINB   =$16
; podemos usar de R16 a R31 para constantes e instrucciones lógicas como SBCI, SUBI, CPI, ANDI, LDI y ORI
.def tempport =r16
.def count1   =r17          ; instrucciones lógicas
.def count2   =r18
.def   XL     =r26
.def   XH     =r27
.def   YL     =r28
.def   YH     =r29
.def   ZL     =r30
.def   ZH     =r31
;*****************************************************************************************
.cseg                             ; segmento para el código
.org 0
       rjmp   init                ; origen
.org 4
init:  ldi    tempport,$FF
       out    DDRB,tempport       ; configurar puerto B como salida
       ser    tempport
       out    PORTB,tempport      ; poner los pines el puerto B en alto
start: ldi    tempport,$FE        ; pone el bit 0 de tempport en bajo
       out    PORTB,tempport      ; escribe en el puerto
rot_L: rcall  wait                ; espera a un bit
       sec                        ; activa el bit de acarreo
       rol    tempport            ; rota a la izquierda tempport
       out    PORTB,tempport      ; escribe en el puerto
       sbrc   tempport,7          ; salta la siguiente linea si el bit superior es 0
       rjmp   rot_L
rot_R: rcall  wait
       sec                        ; activa el bit de acarreo
       ror    tempport            ; rota a la derecha tempport
       out    PORTB,tempport      ; escribe en el puerto
       sbrc   tempport,0          ; salta la siguiente linea si bit inferior es 0
       rjmp   rot_R
       rjmp   rot_L               ; hacer todo de nuevo
       ; ***************subrutina de espera ****************************
wait:  ldi    count1,200          ; carga el registro count1 con el número 200
d1:    ldi    count2,200          ; cara count2 con el número 200
d2:    dec    count2              ; decrementa count2
       brne   d2                  ; lazo si no es cero
       dec    count1              ; decrementa count1 si count2 es cero
       brne   d1                  ; ejecuta de nuevo el lazo si count2 no es cero
       ret
```

Figura 11.2 *Programa ejemplo*

INDICE GENERAL

1. Conceptos básicos de programación 1
Introducción 2
Fundamentos de programación 2
Lenguajes de programación 4
El diagrama de bloques o de flujo 6
De la computadora al microcontrolador 8

2. Sistemas de numeración en la programación de microcontroladores 9
Estructura de los datos 10
Bit 14
Nibble 14
BCD 14
Byte 14
Word 14
Rango de los números binarios 14
Operaciones básicas 15
La suma de dígitos binarios 15
Suma de números binarios 15
Números binarios negativos 16
Ejemplo 16
Resta de números binarios 16

3. Programción de microcontroladores 17
Control o sistema embebido 18
Archivos generados por el programa ensamblador 19
Términos usuales 20
Estructura de un programa escrito en lenguaje ensamblador 20
Tipos de línea que se pueden incluir en un programa .. 21
El entorno de programación 22
El contador de programa 22
Conjunto (*set*) de instrucciones 23
Instrucciones de desplazamiento de *bits* 25
Aplicación del desplazamiento circular 26
El registro de estado 26
Modos de direccionamiento 27

4. Modelo de programación para los microcntroladores PIC 29
Los elementos 31
La aquitectura del PIC 16F84 (La visión del programador) 31
El registro de trabajo W 31
El registro de estado (*status register*) 31
La memoria de programa 32
La memoria de datos en RAM 33
Los registros de propósito especial 33
Los registros de propósito general 33
Los registros de pila (*stack*) 34
La memoria EEPROM de datos 34
Subprogramas 34
Funcionamiento de los registros de pila del microcontrolador 35
Puertos 36
El puerto A 36
El puerto B 36
Configuración de los puertos A y B 36

5. Herramientas de *software* para el desarrollo de aplicaciones basadas en microcntroladores PIC 37
Elementos orientados al *hardware* 38
Elementos basados en *software* 38
El editor de texto 38
El ensamblador 39
El ensamblador MPASM de Microchip 39
Archivos asociados al ensamblador MPASM 40
Código absoluto y código objeto 40
Ejecución del programa MPASM desde la línea de comandos 40
Directivas 42
Directivas de control 42
Directivas condicionales 43
Directivas orientadas a los datos 43
Directivas de control de listado 44
Directivas orientadas al archivo objeto 44
Macros 44
Sintaxis de un macro 45
Directivas orientadas a los macros 45
Uso de los macros 45
Mensajes generados por el ensamblador (*messages*) 45
Algunos mensajes básicos 45
Advertencias (*warnings*) detectadas por el ensamblador 46
Algunas advertencias básicas 46
Errores (*errors*) detectados por el ensamblador 46
Algunos errores básicos 46
Operadores, operandos y expresiones 47
Cadenas (*strings*) 47
La secuencia de códigos de escape ANSI'C 47
Sistemas numéricos 47
Tabla de códigos ASCII 48

6. Ejercicios de programación básica con microcontroladores PIC 49
Ejercicio No. 1
Encendido de un diodo LED 50
Circuito basado en un microcontrolador 50
Montaje del circuito utilizando un *protoboard* 50
Montaje del circuito utilizando un entrenador 50
Desarrollo del proyecto 50
Registros utilizados 51
Diseño del programa usando el lenguaje ensamblador . 51

Transferencia de información hacia el puerto B 52
Almacenamiento del dato 0x01 en el puerto B 52
Configuración del puerto B 52
Detención del flujo de ejecución 53
Directivas de control 53
Código fuente del proyecto 54
Ensamblando el programa 56
Grabando el programa en el microcontrolador 57
Haciendo espacio al vector de interrupciones 58
Diagrama de flujo del proyecto 59
Instrucciones utilizadas en esta lección 59
Ejercicio No. 2
Encender y apagar un diodo LED 59
Propósito general 59
Propósito específico 60
Análisis 60
Estructura repetitiva 61
Precaución 61
Subrutinas 61
Instrucción CALL 62
Instrucción RETURN 62
Estructura de una subrutina 62
Llamados a una subrutina desde un programa 62
Instrucción de soporte 63
Registros de trabajo 63
Control de estructuras repetitivas 63
Estructuras repetitivas anidadas 64
Codificación de un ciclo anidado 64
Listado del programa 64
Conclusión 66
Ejercicio No. 3
Lectura de una interrupción externa (INT) 66
Descripción del programa 67
Ejercicio No. 4
Generación de señales digitales a través del TMR0 70
Ejercicio No. 5
Contando eventos externos 71
Ejercicio No. 6
Generación de señales mediante el TMR0 e interrupciones 73

7. Ejercicios avanzados de programción con microcontroladores PIC 77
Ejercicio No. 7
Conexión de memorias seriales al PIC 78
Desarrollo del ejercicio 80
Ejercicio No. 8
Manejo de un módulo LCD 84
Módulos de cristal líquido ó LCD 84
Interfaz con microcontrolador a 8 *bits* 86

8. Ejercicios con microcontroladores Motorola 91
Nomenclatura 97
Ejercicio No. 9
Energizar un diodo LED 97
Ejercicio No. 10
Encender y apagar un diodo LED 101
Ejercicio No. 11
Multiplexaje de indicadores (*displays*) de siete segmentos 105
Ejercicio No. 12
Manejo del conjunto de instrucciones 113
Modos de direccionamiento 113
Instrucciones de salto 117
Instrucciones varias y manipulación del CCR 121

9. Ejercicios de programción con el BASIC Stamp 125
Desarrollo de programas 126
Ejercicio No. 13
Encender y apagar un diodo LED 128
Ejercicio No. 14
Controlando el diodo LED con un pulsador 130
Ejercicio No. 15
Secuenciador con diodos LED 131
Ejercicio No. 16
Transmisión de datos seriales vía RS-232 132
Ejercicio No. 17
Recepción de datos seriales vía RS-232 134
Ejercicio No. 18
Comunicación serial sincrónica 135

10. Programación de microcontroladores INTEL 137
Modos de direccionamiento 138
Ciclo de instrucción 138
Tipos de instrucciones 139
Instrucciones aritméticas 140
Instrucciones lógicas 140
Instrucciones de transferencia de datos 140
Instrucciones booleanas 141
Instrucciones de salto 141
Modelo de programación 143
Ejercicio No. 19
Estructura condicional 143
Ejercicio No. 20
Retardo por *software* 144
Ejercicio No. 21
Subrutinas 145
Ejercicio No. 22
Manejo de la pila y de las rotaciones 146

11. Programación de microcontroladores ATMEL 149
Manejo de registros 150
Registros apuntadores 150
Recomendaciones para el uso de registros 152
Los puertos en los microcontroladores AVR 152
La memoria SRAM 152
¿Para qué podemos usar la SRAM? 153
¿Cómo utilizar la SRAM? 153
Utilización de la SRAM como pila 153
Definiendo la SRAM como pila 154
Utilización de la pila 154
Introducción a la programación de microcontroladores AVR 155
Estado de *reset* 155
Ejecución lineal de programa y saltos 156
Temporización durante la ejecución de un programa ... 156
Macros 156
Subrutinas 156
Interrupciones 157

Obtenga su certificado de entrenamiento en sólo 40 semanas

Al final del curso se publicará un completo cuestionario para la evaluación de sus conocimientos.

Al contestarlo correctamente, usted obtendrá un certificado de estudios expedido por CEKIT S.A.

Capacítese en Microcontroladores y forme parte activa de la economía del nuevo milenio en su profesión o su hoobie, realizando este fácil y rápido...

Certificado de Aptitud

CEKIT
COMPAÑIA EDITORIAL TECNOLOGICA

Certifica que:

Marcelo Andres Zamora Ugalde

Completó satisfactoriamente el curso sobre

Microcontroladores

incluyendo Teoría, Programación y Diseño.

Marcelo Andres Zamora Ug

CEKIT
Le ayudamos a construir su futuro

Director Técnico

Marcelo Andres Zamora Ug

Fecha

Ciuda

CURSO PRACTICO SOBRE

MICROCONTROLADORES

Otro producto con la calidad y garantía de

Es de hacer notar que el presente certificado da idea de haber cumplido con los conocimientos básicos de la teoría y práctica del curso.